BASIC

ELEMENTAL Y SISTEMATICO

EL BASIC DE TODOS LOS MICROORDENADORES
DEL MUNDO DE HOY

Por:
CARLOS BASTOS NOREÑA

BASIC

ELEMENTAL Y SISTEMATICO

Edita: M.ª Dolores Rocamora Valls.
Impreso en: Gráficas 82 - Navarra, 15 - 28039 MADRID
Depósito Legal: M-34847-1985.
I.S.B.N.: 84-398-5040-9
Printed in Spain

Autor:

CARLOS BASTOS NOREÑA
Doctor Ingeniero Industrial (Madrid).
Analista de Sistemas por NCC.
Gerente de Bastos Microinformática.
García de la Parra, 2. ARAVACA (Madrid)
Teléf. (91) 207 99 77

Dibuja: Sebastián de la Serna.

Dedicatoria: A Iñaki Bastos
el de Cabo de Palos
el investigador

PREAMBULO

Esta colección va dirigida específicamente a quienes sienten la necesidad de profundizar en un BASIC que ya les es familiar en mayor o menor grado; pero hemos procurado que sirva también al que ahora se inicia en las lides de la programación, sin otro bagaje informático que el de unos conocimientos rudimentarios de lo que es un ordenador.

Para unos o para otros, nuestro empeño ha sido **llegar al final** de lo que el Basic puede dar de sí: Pretencioso es nuestro objetivo. En él incluimos la posibilidad de dar satisfacción a aquellos que sabiéndose necesitados de un microordenador (para su empresa, para su departamento, para su hogar) estén dispuestos a hacer el **esfuerzo necesario para no tener que depender de nadie.** Para ser autosuficientes en la elaboración de sus propios programas.

Es fama que la Informática, desde su nacimiento, está reñida con la Pedagogia. El que dice saber de ordenadores, se nos aparece como un ser de distinta especie, híbrido de oráculo y esfinge, y que habla un jerga que sólo unos pocos iniciados entienden.

Por todas partes se respira ya la necesidad de desmitificar la informática. Y el hacerlo es urgente y necesario aún a riesgo de sufrir el desprecio o la burla de los puritanos. Si hay que elegir, mejor siempre la claridad que el rigor.

La Informática ha venido a continuar algo que está en la misma raiz de la evolución cósmica. Del átomo al diamante, de la célula al hombre la evolución ascendente del Cosmos ha sido posible gracias a una **información que se transmite** de generación en generación.

Esa «información transmitida» (secreto último del sentido positivo de la evolución) alcanza su paroxismo con la irrupción del hombre en la tierra. Porque la información ya no se transmite sólo por vía genética, vertical, generacional. Horizontalmente, el torrente de inter-informaciones es inconmensurable. E incesante es la transmi-

sión de conocimientos y experiencias entre individuos y entre comunidades.

Todo permanece. Nada valioso se pierde. No hizo falta más que un solo Prometeo. El fuego se descubrió una vez. Como la rueda, como el arado, como la energía nuclear. Transmisión implacable de una Información inverosímilmente rica, variada y multiforme.

Y he aquí que cuando parece que el hombre está llegando al límite de sus posibilidades en el dominio de la evolución, ve nacer de sus propias manos un instrumento capaz de elaborar, procesar y transmitir datos a una velocidad mil veces superior a la conseguida con cualquier intrumento precedente.

Se trata de una herramienta tan poderosa que hasta le puede liberar de todo trabajo no creativo. Es como una prolongación de sí mismo. Algo que está en la línea de lo que la misma naturaleza, lenta y laboriosamente, ha venido haciendo desde el principio de los tiempos: dominar la información, mantenerla y transmitirla.

El hombre ya no es tanto sujeto paciente de la evolución como sujeto agente de la misma. Puede, con un leve golpe de timón, modifical la ruta del devenir evolutivo. Esta grave responsabilidad rerequiere, en todos los campos del saber y del hacer humanos, disponer de una valiosa herramienta de trabajo. Algo que permita al hombre diponer rápidamente de la información necesaria para que la aventura ascendente, comenzada hace unos cuantos miles de millones de años, continue hoy, llevada por sus manos creadoras.

Y ese instrumento necesario para guardar, investigar, procesar cualquier información, la herramienta que puede suplir al hombre en su deficiente capacidad de memoria y de cálculo, que le lleve a la incorporación del cosmos a sí mismo, no es otro sino el Ordenador. Será su propia libertd la que haga que esta incorporación sea centrípeta y empobrecedora o centrífuga y enriquecedora. Pero esto se escapa ya de los límites de este trabajo.

Henos pues, en presencia de una primera justificación seria de nuestros textos. Sintámonos en casa. Vamos a manejar cosas que guardan una relación directa con lo más íntimo y radical de nuestra propia condición humana.

Así, cuando hablemos de datos, de registros, de archivos, de todas esas cosas que parecen invitar al bostezo, mirémoslas como elementos imprescindibles de una ordenación análoga a la que la propia naturaleza emplea y con la que tan lejos ha podido llegar en su evolución.

Desde ahora y hasta la última página tengamos presente el «un sitio para cada cosa y cada cosa en su sitio», como base de trabajo sin la que nada podríamos hacer. Y si esto nos parece menguada motivación, podemos recordar que Agustín de Tagaste definía la Belleza como **el esplendor del Orden.** Y Agustín, en su siglo V, no sabía de informática ni de ordenadores.

El ordenador admite más de una definición, según sea el punto de referencia en que nos encontremos. Un humanista lo definiría seguramente así:

«Ingenio capaz de liberar al hombre de **toda** tarea rutinaria, de **todo** trabajo no creativo.»

Poniéndonos más a ras de tierra, ésta puede ser una definición bastante real de lo que es un ordenador:

«Máquina capaz de recoger unas **entradas** (leer unos datos de entrada) y producir con ellos unas **salidas** (dar una información de salida) gracias a un **proceso** (interpretación y ejecución de un programa). El **programa** ha de ser algo incorporado a la máquina **desde el exterior.** Los resultado a obtener deben poder ser tan variables como lo desee el creador del programa, sin más limitación que la lógica, el tiempo, el espacio y la propia capacidad física de la máquina.»

La lavadora automática, el automóvil, la calculadora y el ajedrez electrónico no son ordenadores: obedecen a un programa interno, inmutable con el «hardware», con las propias tripas del mecanismo.

Cualquier máquina que no sea un ordenador, por muy sofisticada, ingeniosa y complicada que sea, será en todo caso algo capaz de realizar una tarea (o varias relacionadas entre sí). Un ordenador en cambio, puede obedecer las instrucciones de programas tan dispares como llevar la contabilidad analítica de una empresa, calcular la estructura de una torre de refrigeración de una central nuclear o gobernar el vuelo de un cohete espacial.

Los microordenadores no aparecieron en el mercado en un número significativo hasta la frontera de las décadas 70-80. Sin embargo, los microprocesadores en los que aquellos se basaban, nacieron en 1971 cuando Ted Hoff inventó el INTEL 4004, precursor de la supervedette INTEL 8080 (1973).

Y fue en 1976 cuando Zilog introdujo en el mercado el Z80 y en 1978 el Z8000. Intel contraatacó a su vez con el 8088 y 8086, mi-

croprocesadores que forman parte del 90% de los microordenadores de hoy (1985-1986).

En realidad, y como sucede en todo lo nuevo, hay dos fechas incorporadas a cualquier ingenio informático. La primera es la que corresponde al **anuncio** de un producto y la segunda (1 ó 2 años después) es la de la **aparición** confirmada de ese producto a disposición del usuario.

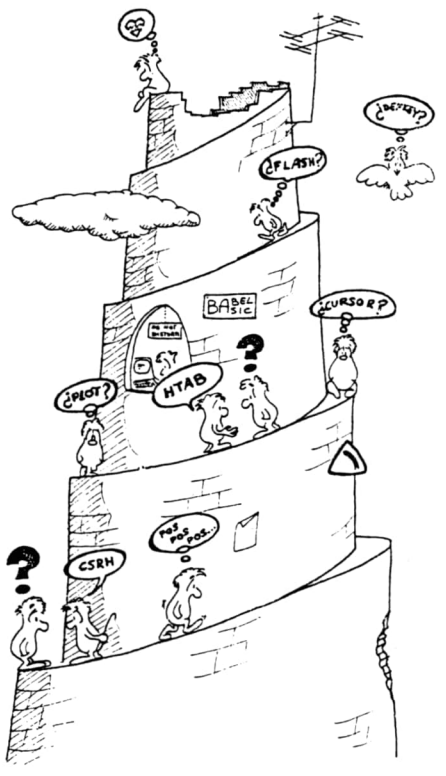
A este respecto tenemos que decir algo de verdadera importancia.

La informática avanza deprisa. El presunto usuario-adquirente de microordenador, debe procurar estar a la última, estamos de acuerdo; pero con tiento: que esa obsesión por «lo más nuevo» no sea desmedida.

Cuando en el mercado aparece un nuevo —revolucionario— modelo de ordenador deberá transcurrir bastante tiempo hasta que ese producto (hardware) esté probado (y bien probado), y hasta que el «alto software» que le acompaña esté libre de impurezas. La elección del sistema será pues, como en todo, el viejo compromiso entre «progreso» y «seguridad». Muy a menudo en informática —tal vez con un rigor mayor que en otros campos— es bueno hacer uso (sin abuso) del «más vale malo conocido que bueno por conocer».

PARTE PRIMERA. PROGRAMACION ELEMENTAL

CAPITULO I.— FUNDAMENTOS DEL BASIC (I)



Empezamos con una serie de definiciones y aclaraciones cuya finalidad puede no ser bien comprendida por el momento. Recomendamos al lector que se sepa impaciente, eche un rápido vistazo sobre estas primeras páginas, y vaya directamente donde comienza la programación propiamente dicha (Capítulo 2 de esta Primera Parte). Y a los que ya les sea familiar el Basic de Microsoft y quieran ir al grano, les sugerimos se sitúen en la Segunda Parte. Tanto unos como otros, pueden, posteriormente, retroceder y utilizar estos primeros capítulos como puro texto de consulta.

LOS BASIC Y EL BASIC DE MICROSOFT

Todo microordenador —y muchos miniordenadores— tienen al BASIC como su principal lenguaje de programación. Pero así como un mismo idioma puede tener varios dialectos, otro tanto ocurre con el lenguaje que nos ocupa. No existe *el BASIC* sino *LOS BASIC.*

Cada fabricante de ordenadores parece recrearse en ponerle florituras a *su* BASIC para desesperación de profesionales y usuarios.

Si comparamos el BUSINESS BASIC de Nixdorf con el BASIC-250 de Hewlett-Packard o con el BB de Basic Four, observaremos que junto a muchas similitudes, hay también abismales diferencias.

Por fortuna, frente a estas tendencias disgregadoras hay otras que empujan en sentido contrario, procurando la máxima compatibilidad entre ordenadores de distintas razas (marcas, tamaños y países de origen).

En el caso del BASIC, una firma de software americana llamada Microsoft se ha erigido, desde 1976-77 en líder de esa unificación, creando el MBASIC también conocido por BASIC-86 y MSBASIC en ordenadores a 16 bits. Este BASIC opera bajo CP/M, MSDOS, TURBO-DOS y otros supervisores logrando así hacerse prácticamente universal ya que estos sistemas operativos son implantables en la mayor parte de los microordenadores del mercado de hoy.

En lo que sigue, y mientras no digamos nada en contra, nos ceñiremos siempre al BASIC de Microsoft. Este será nuestro BASIC por defecto.

POR DEFECTO

Con toda intención hemos escrito en el párrafo anterior BASIC *por defecto* a sabiendas de que muchos lectores van a tacharnos con razón, de francotiradores del castellano. Nos explicamos.

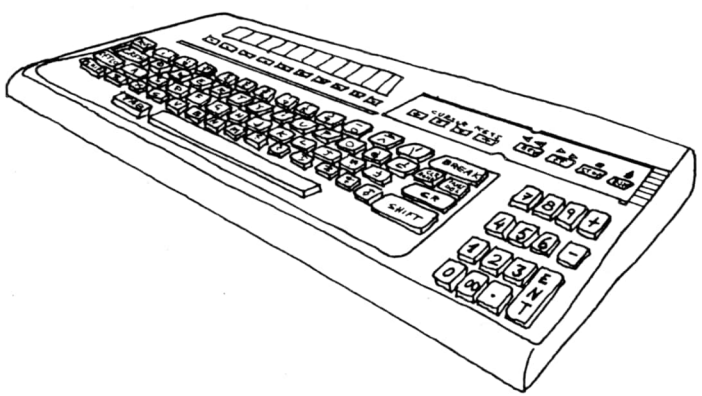
Todo el secreto de la informática se asienta sobre la base de una continua obsesión por lo más *rápido*, lo más *cómodo* o lo que menos *espacio* ocupa. En una palabra, y en un sentido amplio, por lo más *económico.* En base a esto, el informático se enfrenta constantemente, a todos los niveles, con situaciones en las que le gustaría que el ordenador «adivinara» lo que pretende, con el vergonzoso propósito de trabajar menos. Pues bien, no. El ordenador no va a adivinar nada. Pero lo que sí puede hacer es «predisponerse» para la situación que se presente con más frecuencia. Más aún: esta «predisposición» es a menudo programable, lo cual puede ser de una gran ayuda para el programador. Veamos.

En inglés se habla del «default value» o del «default drive» cuando se refiere al «valor sobrentendido» y a la «unidad de discos sobreentendida» respectivamente. Traduciremos «default» por «por defecto», mal que le pese a nuestra Real Academia de la Lengua.

Así pues —y si nos hemos extendido tanto en esto es porque el tema lo merece—, en las mil ocasiones en que de ahora en adelante aparezca la expresión «por defecto», querremos decir «que se sobreentiende mientras no lo cambiemos».

Por ejemplo, si decimos que X vale 0 por defecto ello significa exactamente que su valor original es 0 y que seguirá siéndolo mientras nadie le asigne otro. O —y terminamos— si queremos jugar con la trigonometría y nos dicen que la unidad angular por defecto es el radián, eso quiere decir que para trabajar en grados sexagesimales tendremos que decírselo expresamente porque si no, la máquina operará indefectiblemente en radianes.

EL TECLADO



Antes de meternos en harina, conviene explicar —o recordar— algunas peculiaridades del teclado de los ordenadores.

La tecla más importante, la más utilizada, la única que suele estar repetida (en la zona *máquina de escribir* y en la zona *calculadora*), la que además presenta un tamaño mayor, es la denominada *CR,* o ENTER, o ENT, o RETURN o INTRO.

Nos quedaremos con CR por más frecuente. CR, iniciales de «Carriage Return» («Retorno de Carro») quiere recordar a la tecla (o la palanca) que a tal fin tenían y tienen las máquinas de escribir de todos los tiempos.

Cuando en lo sucesivo, digamos simplemente *CR* o <CR>, se traducirá por «apriétese la tecla marcada o CR, o cualquiera de sus equivalentes (ENT, ENTER, etc.)».

Normalmente todos los ordenadores tienen dos teclados numéricos: no hay ninguna diferencia entre ellos. Desapasionadamente recomendamos el uso del que está a la derecha, tipo calculadora, por ser más útil y rápido, una vez que el operador o programador se acostumbra a él. Esta recomendación va dirigida especialmente a los/las mecanógrafos/as ya viciados/as por el uso del teclado numérico típico de la máquina de escribir.

Cada tecla —también como las máquinas de escribir— dispone de mayúsculas y minúsculas. Y las alfabéticas (y alguna más) tienen una tercera alternativa —al menos— que vienen dada por el empleo de la importantísima tecla de *CONTROL.* Esta tecla suele presentarse con las letras CTRL. En ocasiones aparece con las letras ALT (de alternativa), GRAPH (de gráficos), etc.

Si, manteniendo apretada esta tecla de control, se pulsa una letra, el efecto de nuestra acción puede ser muy variado.

Cuando, en lo que sigue, escribamos *CONTROL-C* o *CONTROL-H,* estaremos diciendo lo expresado antes: que se *mantenga* apretada la tecla de control mientras se pulsa la letra en cuestión, soltando luego ambas.

AL final del capítulo siguiente damos un resumen de las funciones que realizan las teclas susceptibles de ser utilizadas con la de control.

PUESTA EN MARCHA

En todos los quehaceres humanos, cuando se es enteramente novato, lo primero que uno dice (o piensa, si no se atreve a decirlo) es poco más o menos esto:

—Mire Vd., yo de esto no sé absolutamente nada, así que, por favor, explíquemelo todo desde el principio.

De acuerdo. Vamos allá. Está Vd. ante una máquina que tiene, al menos, un teclado, una pantalla, una o dos ranuras con aspecto de buzón, y tal vez otras cosas que de momento no nos importan. Disponemos de una toma de corriente en la pared. Enchufamos la máquina como quien enchufa un secador de pelo. No pasa nada todavía.

La máquina, en algún sitio exprofeso incómodo (para dificultar el accidente —que puede ser dramático— de que se *apague* el sistema en pleno trabajo), probablemente en su parte trasera, tiene un interruptor vulgar. Damos al interruptor. (A lo mejor la pantalla se *enciende* aparte. Si es así, lo hacemos.)

Algo se mueve. Leves susurros, algún chasquido. Quizá se encienda alguna pequeña luz. En la pantalla tiene que aparecer algún mensáje por lacónico que sea. Seguramente —¡ay!—, en inglés.

Al poner en marcha el sistema (que es precisamente lo que acabamos de hacer), se activa un chip ROM, una especie de «programa en conserva», invariable, intocable.

Ese ROM contiene un programita cuya única misión es abrir las compuertas que propicien el volcado del Sistema Operativo desde la memoria externa (desde el diskette, disco o cassette) hasta la memoria interna. Bástenos recordar el estado de virginidad total en que se encuentra la memoria interna cuando la máquina está apagada. Y que el Sistema Operativo *tiene que estar presente siempre* en memoria interna para que todo funcione.

En pocos segundos, el ordenador habrá leído de la memoria externa el conjunto de instrucciones que componen el sistema operativo (que, por importante que sea, no pasa de ser un programa) y las habrá situado en algún lugar de la memoria interna, protegiendo al propio tiempo esa zona de la memoria para que a nadie se le ocurra invadirla.

A esta operación de arranque, los informáticos, siempre tan suyos, le dan el bonito nombre de «bootstrap» o «boot» o «hacer IPL» (iniciales de «Initial Program Loading» = Carga del Programa Inicial).

Pero atención: el hecho de que el S.O. (el Sistema Operativo, que desde ahora designaremos por sus iniciales) esté en mermoria, no significa que el disco (la memoria externa, en general) haya perdido nada. El S.O. estará en ambas memorias, aunque sólo «la copia» que acaba de pasar a la interna es la que estará en condiciones de manejar todo el cotarro.

Y como hemos dicho que nos vamos a referir siempre al BASIC de Microsoft, supondremos que el S.O. es, por ejemplo, el CP/M en ordenadores a 8 bits o el MS/DOS en ordenadores a 16 bits. En esas condiciones, al final, y cualesquiera que sean los mensajes que hayan podido aparecer en la pantalla, veremos en ella exactamente una *A* seguida de un angulito. Así:

A>

A la derecha del angulito veremos el llamado CURSOR, parpadeante o no, como un cuadradito o como un guión, da lo mismo. Si ahora pulsamos una techa, aparecerá en donde estaba el cursor el carácter que hayamos apretado, desplazándose enseguida el cursor a la derecha, y así sucesivamente; como si la pantalla remedase el papel de una máquina de escribir.

La *A* significa que el diskette que esté en el «drive» A (en el buzón A) es la memoria externa que la carga del sistema ha elegido *por defecto*. Es decir, que cualquier cosa que pidamos ahora a la memoria externa, aquélla será buscada en el diskette A. Mientras no cambiemos de disco por defecto, será el A el que mande. ¿Y cuál es el disco o diskette o buzón o drive A? Pues el de la izquierda o el de arriba, según marca o modelo. Puede ser el otro (o uno de los otros si tiene más de dos). Si le interesa asegurarse, basta con que escriba con el teclado lo que quiera, pulsando la tecla CR al final. Verá que se enciende la lucecita roja de uno de los buzones: la que se haya encendido le indicará cuál es el diskette por defecto

(dócilmente está buscando lo que Vd. haya escrito, por peregrino que sea).

Retrocedamos un poco. Decíamos que al encender la máquina nos saldría una *A* seguida de un angulito. Y eso quiere decir que el sistema operativo está en la memoria interna. Y que el disco por defecto es el *A*. Pero puede ser que el disco instalado no disponga del S.O. En la pantalla nos aparecerá un mensaje que ponga de manifiesto nuestro despiste, instándonos a que pongamos el disco correcto. Nada serio. Se cambia el diskette, y listo.

También puede suceder que se nos haya olvidado poner el diskette, o que éste no esté formateado, o que haya un conector suelto, o que exista cualquier otra deficiencia. Relájese. Nada es grave. Apriete las conexiones, instale el disco correcto y enseguida tendrá Vd. en su pantalla la letra y el ángulo.

MODOS DE OPERACION

Se dice que el operador está en *Modo Control* cuando el control (el mando, la iniciativa) está en manos de aquél (de Vd. en este caso, con su A>. Ahora puede Vd. hacer muchas cosas. Pero no puede Vd. ponerse a programar, así directamente. Porque para poder hacerlo, necesitamos indefectiblemente que en memoria interna también esté presente un programa intérprete. Un programa de bastante categoría que *traduzca* lo que Vd. va a decirle en Basic, que es algo ininteligible para el microprocesador, en instrucciones directamente comprensibles y ejecutables por éste.

Tecleemos, por ejemplo, *DIR* (de «Directorio», índice) y pulsemos la tecla CR. Lo que vemos que aparece en pantalla es sencillamente la lista de lo que contiene el diskette por defecto. Si escribimos *DIR B:* y CR, saldrá el contenido del diskette que se encuentre en el buzón B (el disco por defecto seguirá siendo el A). Si queremos que el control pase al disco B, bastará con que escribamos *B:* y CR.

Si en la lista de ficheros-programas de uno y otro diskette no vemos que haya ninguno que se llame *MBASIC.COM, MSBASIC.EXE,* o parecido, mucho nos tememos que nos vamos a quedar con las ganas de programar en BASIC, a menos que (todo puede ocurrir en esta Informática del Señor) algún bromista haya cambiado el nombre del programa-intérprete de BASIC, cosa muy fácil de conseguir.

En el supuesto de que todo esté como es debido, y con el cursor a la derecha del angulito de marras, escribamos *MBASIC* y CR.

En cosa de pocos segundos veremos aparecer en la pantalla una ristra de mensajes (que leeremos con frío estoicismo) con un Ok al final. El intérprete está ya en memoria interna. Ahora sí que estamos en condiciones de programar. El cursor, ahora debajo del Ok, está ahí pacientemente a la espera de que escribamos algo. Algo en BASIC.

Seguimos con el mando del aparato. Seguimos en *Modo Control.* Pero con un sutil diferencia con el anterior. Antes, con la *A>*, podíamos ejecutar cualquier comando del sistema operativo (uno de los cuales es el MBASIC). Ahora no. Ahora estamos inmersos en el BASIC. Hemos perdido altura aunque con la ventaja de que el sistema va a entender todo cuanto le digamos en BASIC. Si queremos volver al sistema, escribiremos *SYSTEM* seguido de CR y ya está.

Estamos en BASIC. Hay un Ok encima del cursor. Vamos a fijar nuestra atención en el penúltimo mensaje de los que aparecen al cargar el intérprete. Dirá algo así como:

32554 Bytes free

Ello quiere decir que disponemos de esos bytes para programar. ¿Y qué ha pasado con el resto hasta 64 K (65535 bytes) que decía tener la memoria interna de nuestra máquina? Es la cuenta de la vieja: el Sistema Operativo se lleva de 6.000 a 10.000 bytes. El intérprete BASIC ocupa de 20.000 a 25.000 bytes (seria cifra). Las cuentas salen, nadie ha hecho trampa. Bueno, sí, algunas veces la publicidad de las máquinas hacen una pequeña trampa. Dicen, por ejemplo, «64 K libres de usuario», cuando en realidad, entre el S.O. y el intérprete se quedan con la mitad de esa cifra. Más correcto sería anunciar «64 K en RAM».

Conocemos pues, dos formas de *Modo Control:* uno dentro del sistema y el otro dentro del Basic. Instalémonos ya en este último. Dispondremos de la iniciativa hasta que no la perdamos. Esto es de perogrullo. Y si la perdemos es que voluntariamente así lo hemos querido. Esto sucede siempre que ponemos en ejecución un programa.

Pero a menos que el programa se esté ejecutando «compilado» (ver más adelante), siempre hay un procedimiento rápido y sencillo de «abortar» (interrumpir) la ejecución del programa y recuperar así el modo control. Esta práctica, del todo inofensiva en la mayoría de los casos, puede conllevar algún peligro en ocasiones, como ya veremos al tratar del manejo de ficheros.

Si deseamos actuar sobre un programa que se está ejecutando (es decir, que está sumisamente haciendo lo que le hemos dicho que haga), de nada nos valdrá aporrear el teclado. Nadie nos hará caso. Estamos exagerando. Porque en realidad sí que hay un modo de escaparse del programa y recuperar el control absoluto. Es suficiente pulsar CONTROL-C. Veremos entonces aparecer en pantalla algo semejante a «Break in 210», con lo que el ordenador, con un cierto matiz de reproche, nos está diciendo que su precioso trabajo ha quedado interrumpido (roto) en ese número de línea.

Tenemos entonces, por ahora, *tres modos* de encontrarse el ordenador respecto al operador:

- Modo Control (Sistema).
- Modo Control (Basic).
- Modo Programa en ejecución.

En este último podemos incluir el caso de que el programa que se está ejecutando no sea un programa Basic y lo hayamos llamado por consiguiente, no desde el Basic, sino desde el propio sistema. Ejemplo: copia de discos. Programa con el que grabamos en un diskette el contenido completo de otro. Este tipo de programas se ejecutan directamente desde el Sistema Operativo. No son Basic. Ni que decir tiene que cuando un programa de esta naturaleza se está ejecutando, un intento de abortar el programa sería cometer un verdadero delito informático.

En lo sucesivo, nuestro Modo Control «por defecto» será el que disponemos cuando estamos dentro del Basic.

Dentro de este modo control Basic podemos a su vez actuar de dos maneras:

- Introducir líneas de programa: todo cuanto escribamos precedido de un número entra en la memoria interna con la categoría de *línea de programa*, como algo que posteriormente se quiere ejecutar, de acuerdo con las normas que enseguida vamos a ver.

- Escribir algo que no esté precedido por un número, es decir, utilizar el Basic en *modo directo.* Como una calculadora, aunque con muchas más posibilidades. Volveremos sobre esto.

Existe un cuarto e importante Modo Operativo —situación del operador respecto al ordenador— que merece rancho aparte. No dis-

ponemos del control de la máquina (no podemos hacer lo que nos venga en gana) ni tampoco está funcionando ningún programa. Se trata del Modo Edición o Modo Edit. Más claro sería para nosotros llamarlo «Modo Corrección» o, si se nos perdona el pleonasmo, «Modo Modificación». Ahora, tan sólo lo mencionamos. De él trataremos en su momento.

DEFINICIONES

Ardua tarea la nuestra. Orden, mandato, comando, directiva, declaración, instrucción, sentencia, función... Toda una jerga desparramada por los textos, que está pidiendo a gritos una llamada al orden. Una singular anarquía reina en el léxico informático. Y esa anarquía resulta convertirse en la práctica en un ingrediente valioso para quienes les interesa mantener vivo el mito de los ordenadores. Cada firma llama a las cosas —aun las más triviales— con nombres muy suyos, a menudo rimbombantes, tabú para los no iniciados. Ni aun los que escriben gruesos diccionarios especializados parecen ponerse de acuerdo. O es que tal vez, es la propia juventud de la Informática la causa de este embrollo.

Conscientes de ello, nosotros vamos a procurar juntarnos a la mayoría, advirtiendo al lector que no se extrañe si en otros textos encuentra alguna contradicción con el nuestro y aconsejándole de paso que lea las definiciones que damos a continuación con una cierta dosis de saludable escepticismo.

DIRECTIVA

Toda palabra inteligible para el intérprete Basic que lleve a una acción. Hay tres tipos de directivas:

* COMANDO (u ORDEN o MANDATO)
* SENTENCIA
* FUNCION

COMANDO, ORDEN, MANDATO

Sinónimos. Nos quedamos con el primero, no sin una cierta pena. Es el más utilizado y el menos castellano.

Llamamos así a toda palabra (verbo transitivo siempre) que acompañada o no de otra u otras, y siempre seguida de un CR, produzca un resultado que pueda afectar a todo un programa o una parte significativa del mismo.

Ejemplos: `RUN, LOAD, MERGE, RENUM, LIST`

Los comandos no suelen ir *dentro* de un programa aunque la mayor parte de ellos pueden hacerlo.

SENTENCIA

Palabra o excepcionalmente conjunto de palabras (verbos casi siempre) que, acompañadas o no de otras, produzcan una acción en sentido general.

Lo único que de verdad distingue a la sentencia del comando es que la sentencia se queda dentro de la instrucción en la que interviene, no afecta (o no va dirigida) a un programa completo o a una parte del mismo como sucede con el comando.

Ejemplos: `PRINT, GOTO, GOSUB, FOR ... TO ..,IF, DATA`

INSTRUCCION

Es el conjunto de una sentencia seguida de los elementos necesarios para llevar a cabo una acción. Puede ser autosuficiente, es decir, tener sentido por sí sola, o depender de otra instrucción.

Ejemplos:
```
IF A>B GOTO 1200
SWAP X,Y
ON N GOTO 300,350,400
PRINT "ALBERTO TIENE ";N;" PESETAS"
```

LINEA DE INSTRUCCIONES

Conjunto formado por un número natural cualquiera comprendido entre el 0 y el 65529, seguido de una o más instrucciones. Cuando contiene más de una instrucción éstas irán separadas por el signo «:» (dos puntos).

Ejemplos:
```
325 IF MES>31 OR MES<1 THEN 140 ELSE 400
610 PRINT: GOSUB 1100: GOSUB 20: READ A:ON A GOTO
    300,500
```

El primer elemento es una línea de una sola —aunque compleja— instrucción. El segundo contiene 5 instrucciones, todas ellas pertenecientes a la línea 610.

PROGRAMA

Conjunto de líneas de instrucciones, relacionadas entre sí de tal manera que su ejecución produzca los resultados previstos por el programador. Puede ser tan sencillo como conseguir una relación

de los números pares del 2 al 40 o tan complicado como sacar una estadística de consumo de zanahoria en el Mediterráneo Oriental durante los meses sin R en la década de los 80.

Ejemplo: los innumerables que aparecen en las páginas sucesivas de este texto.

APLICACION

Conjunto de Programas referidos a una misma materia.

Ejemplos: Contabilidad. Nóminas. Gestión de Almacén.

FUNCION

Palabra que, acompañada casi siempre de otra llamada «argumento» (que va entre paréntesis), produce una modificación de éste. Esa modificación está definida precisamente por la Función de que se trate.

Ejemplos: `INT, VAL, SIN, POS`

CAPITULO 2.— FUNDAMENTOS DEL BASIC (II)

En este capítulo vamos a estudiar:

— Constantes y Variables en toda su variedad, incluyendo vectores y matrices.

— Operadores: aritméticos, relacionales y lógicos.

— Otras consideraciones de interés, como son:
  — Compilación e interpretación.
  — Numeración de los programas.
  — Corrección de un programa (modo EDIT).
  — Caracteres de Control.

CONSTANTES Y VARIABLES

CONSTANTES NUMERICAS

Algo tan trivial como los números en cualquiera de sus formas.

Ejemplos: 4, -13, 45, 530000000.

Pero esto, en efecto tan trivial, se nos va a complicar un poco. Y no por el aquel de la magia de los números (que la tienen, como todo el mundo sabe), sino por una cuestión de simple economía. En cualquier programa, en cualquier aplicación, los datos numéricos proliferan como hongos. Si no inventamos algo para sujetarlos, muy pronto nos quedaríamos sin memoria interna y externa. La solución, no sólo de Microsoft, sino de todos cuantos se ocupan de estos menesteres, consiste en establecer *rangos*, categorías de números:

En nuestro Basic definimos tres rangos de números:

1.— Enteros, no superiores a 32767 ni inferiores a -32768

2.— Reales (con decimales si hacen falta), con hasta 6 cifras significativas (7 en el MSBASIC)

3.— Reales con hasta 16 cifras significativas.

A los primeros los llamamos sencillamente *enteros*. Sólo gastan (sólo ocupan) 2 bytes.

A los segundos los llamamos de *simple precisión*. Ocupan 4 bytes.

A los terceros, por último, los llamamos de *doble precisión* y ocupan 8 bytes.

Los de simple y doble precisión, cuando se ven sobrepasados, disparan su «coma flotante» como enseguida veremos.

CONSTANTES OCTALES Y HEXADECIMALES

(De uso muy restringido las incluimos aquí por completar el cuadro de constantes).

Sirven para proporcionar la conversión de números en base ocho y base 16 a base decimal: basta con hacer preceder al número por & *O* (o & a secas) para el octal y por & *H* para el hexadecimal. Por ejemplo, & 44 equivale a 36 en decimal y & 54 se corresponde con 44. Caso singular es el & HFFFF que es el ya conocido 65535 decimal.

CONSTANTES ALFANUMERICAS

Conjunto de caracteres (cualesquiera que éstos sean). Con más frecuencia se les denomina «CADENAS ALFANUMERICAS», o simplemente *CADENAS.* En la elaboración de los programas estas cadenas deben expresarse siempre entrecomilladas, como recurso eficaz para no confundirlas con las variables numéricas como ahora mismo vamos a ver.

Ejemplos: "PACO", "Carlos", "13, Rue del Percebe"

VARIABLES NUMERICAS

Conjunto de letras o de letras y números (también se admite el punto) que representan un valor numérico. Los nombres de estas variables pueden tener una longitud desde 1 hasta 40 caracteres con sólo dos restricciones:

- El primer carácter tiene que ser una letra
- El nombre de la variable no puede ser una *Palabra Reservada*. Son palabras reservadas todos los comandos, sentencias y funciones.

Son *variables* en el sentido de que *podemos* hacer que valgan lo que queramos, de acuerdo con lo que en cada paso del programa nos interese.

Ejemplos: `A, X, PRECIO3, PRECIO.DE.COSTE.1, X123, ZZ`

En un programa puede haber tantos nombres de variables como la memoria interna nos lo permita.

Dada la importancia y cualidades de estas variables vamos a profundizar un tanto en ellas.

Igual que sucedía con las Constantes, hay tres tipos de variables numéricas: enteras, de simple y doble precisión. Si no decimos nada, las variables numéricas adquieren —por defecto— la categoría de *simple precisión* ocupando así 4 bytes.

El rango de las variables puede definirse en el programa, globalmente, como ya se verá cuando entremos de lleno en el BASIC. O individualmente, para lo cual es preciso añadir un pegote al nombre de la variable. Estos pegotes o aditamentos son:

- Para variables enteras: el signo de porcentaje (%)
- Para las de simple precisión: el signo de admiración (!)
- Para variables de doble precisión: la almohadilla (#)

Así, X%, X! y X# son tres variables distintas que ocuparán 2, 4 y 8 bytes respectivamente. Si a la primera se le intenta dar un valor no entero, éste quedará automáticamente truncado al entero más próximo. Y si su valor sobrepasa su capacidad (de -32768 a 32767) aparecerá un mensaje de Overflow (rebasamiento) quedando interrumpido el programa.

Dada la delicadeza que caracteriza a las variables enteras, es muy recomendable pensárselo un poco antes de lanzarse a utilizarlas. Pero en modo alguno debe limitarse su empleo, antes al contrario: siempre que sea posible debe hacerse uso de ellas pues gozan de grandes ventajas como veremos en su momento.

Las variables de simple y doble precisión (llamadas también *reales*) disponen de una manga mucho más ancha: cuando el número de cifras significativas sobrepasa sus propios límites (6 y 16 respectivamente) no se atascan, dan generosamente lo que les es posible dar. Para ello emplean el truco de la mal llamada *coma flotante* (mal llamada porque en realidad se trata de *punto flotante*). Veamos.

Si pretendemos que la variable X! valga el famoso Número de Avogadro (602300000000000000000000), la pantalla, en lugar de lanzar alaridos de protesta, nos dirá que X! es igual al jeroglífico siguiente:

```
6.023E+23
```

Lo cual se leería en castellano «Seis coma cero veintitrés por diez elevado a veintitrés». O sea que la E significa Exponente de 10 y el valor del exponente, con su signo (que puede ser + o - es lo que aparece detrás de la E. El número de la izquierda (6.023), que recibe el bello nombre de *mantisa*, se autoorganiza de modo que siempre presente una parte entera de un dígito.

Si nuestro Número de Avogadro lo asignamos ahora a la variable X —de doble precisión— las cosas serán muy parecidas a lo que acabamos de decir. Sólo cambiará la letra del jeroglífico, que en lugar de ser una E será una D (para recordarnos que estamos en *Doble Precisión*). Oséase: 6.023D+23.

¿Cuál es la diferencia práctica entre una y otra asignación? Con Avogadro, ya vemos que ninguna. Pero si asignamos a X! y a X el valor de 1/7, el primero se convertirá en 1.42857E-01 y el segundo en 1.428571428571428D-01.

El valor más grande que pueden tomar las variables reales es aproximadamente de 1.7E+38, número más bien gigantesco habida cuenta de que el peso de nuestro planeta no llega a 8.0E+25 Kg. Es difícil, pues, que aparezca un Overflow cuando utilizamos variables reales. Si, a pesar de todo ocurriera (por ejemplo si andamos enredando con programas intergalácticos), apareciera un Overflow pero sin error, es decir, sin interrupción del programa. Sólo saltaría el error si —desde luego por haber programado mal— intentásemos dividir un número por cero. El mensaje rezaría entonces «División by cero error».

VARIABLES ALFANUMERICAS

Idéntica definición que la anterior salvo que representan o contienen una cadena alfanumérica, que puede variar con arreglo al programa. Para distinguirlas de las variables numéricas se las hace acabar con el fatídico signo del dólar ($).

Ejemplos: `J$, NOMBRE$, POBLACION$, X$, S12$`

VECTORES Y MATRICES

La denominación más genérica de este importante tipo de variables es el de *Variables Indexadas* y aún más académicamente, el de *Variables Dimensionales* (Uni o Multidimensionales), pero lo cierto es que son más habituales las denominaciones que figuran en el encabezamiento: Vectores y Matrices.

Para entender la razón de ser de este tipo de variables es preciso hacer un paréntesis en nuestro caminar por el Basic.

Imaginémonos la zona de la memoria interna destinada a guardar la información dinámica de nuestro trabajo (la llamada *Area de Datos*) como el enorme casillero de un almacén de ferretería, pongamos por caso.

Ante nosotros se extiende una gran pared, materialmente cubierta por una cuadrícula de anaqueles de —digamos— diez mil receptáculos (para más claridad, imaginaremos que se trata de un cuadrado de 100 × 100 casillas).

Cada una de las casillas o recipientes queda bien localizada por dos números del 1 al 100. Con decir, verbigracia, 47/62, estamos hablando de la casilla que se encuentra en la columna 47, fila 62. No hay otra. La dirección, con esos dos números, queda inequívocamente establecida. (Lo único que quizá pudiera inducir a confusión sería el baile de filas por columnas; pero eso es algo perfectamente definible en un principio).

Imaginémonos ahora que, en lugar de guardar herramientas en ese casillero, vamos a meter números. Uno en cada casilla. Números enteros comprendidos entre el 0 y el 255. (Recordemos que entre esos límites están comprendidos todos los valores que puede tener un octeto o byte. Y para el que no lo recuerde, porque nunca lo haya sabido, que haga ahora una acto de fe, pues más adelante lo vamos a ver.)

Las casillas van a contener entonces, en nuestro caso, bytes, grupos de ocho bits, perfectamente independientes entre sí.

Sin meternos en el procedimiento concreto que el sistema utiliza para saber en qué lugar exacto se encuentra cada cosa, de lo que no cabe la menor duda es que podemos hacer que un grupo determinado de casillas contengan la palabra «RETRUECANO» por malsonante que ésta sea. Bastará con que en diez compartimentos contiguos de nuestro casillero se aloje cada uno de los valores nu-

méricos (código ASCII) que corresponde a cada una de las diez letras que componen RETRUECANO. Con algo más de complicación (por mor de la economía), una cosa parecida sucederá con cualquier valor numérico.

Al final, nuestro casillero se comporta *como si* en lugar de casillas-byte hubiese casillas-dato, sin más que agrupar las primeras para formar las segundas. Más aún, podemos imaginar que cada una de estas casillas-dato dispone de un rótulo con el nombre de la variable. Por ejemplo, si «RETRUECANO» está en R$, buscaríamos en nuestra estantería el cartelito en el que pusiera R$ y, al mirar su contenido veríamos con cierta emoción que allí está nuestro RETRUECANO. Todavía más: si nadie cambia el «valor» de R$ (y sólo el programa puede hacerlo), éste seguirá siendo el mismo mientras no abandonemos el Basic. Esta fidelidad incondicional del ordenador es una de las causas de su proliferación y de su fuerza.

Podemos exigir aún más. Supongamos que queremos disponer de una serie de 7 casillas-dato, que sin dejar de ser independientes, estén de alguna manera relacionados. Pretendemos que en el primer compartimento esté la palabra «LUNES», en el segundo «MARTES»... en el séptimo «DOMINGO». Nos parece mucho lujo emplear una variable numérica distinta para cada día de la semana. Recordemos la obsesión del ordenador por ordenarlo todo con increíble facilidad. Lo mismo podríamos haber hecho con los días de cada mes, con las provincias españolas, con el importe de las ventas de catorce vendedores, etc.

Acabamos de comprender la necesidad de los Vectores (numéricos o alfanuméricos) a los que también se les da el pretencioso nombre de «Series Unidimensionales» y hasta el de «Variables Indexadas de primer orden».

En los vectores se utiliza *un solo* nombre de variable, con un subíndice para cada elemento del vector. Pero como eso del «sub» se adapta mal a los periféricos del ordenador, lo dejamos en «índice». (De ahí lo de «Variables Indexadas»). Y ese índice o número de orden se escribe entre paréntesis y listo.

En nuestro caso, si el nombre del vector que elegimos para guardar los días de la semana es SEM$, SEM$(1) será igual a «LUNES», SEM$(4) será igual a «JUEVES», etc.

De la misma manera podríamos haber hecho con los meses del año: MES$(6) = «JUNIO», MES$(2) = «FEBRERO»...

Análogamente con los días de cada mes. Estaríamos en el caso de un vector numérico: MES(7) = 31, MES(2) = 28, MES(9) = 30...

Esto no tiene sólo la ventaja de la simplificación y el orden. Ocurre en la práctica que no conocemos de antemano en qué mes va a actuar un determinado segmento del programa. Y ese mes (número de mes) que puede ser cualquiera entre 1 y 12, lo tenemos en la variable M. El nombre del mes será MES$(M) y, según sea el valor que tome M, automáticamente tendremos el valor del mes.

Aún podemos ir mucho más lejos. Vamos a suponer que no nos conformamos con una ordenación tan simple, que queremos disponer, no de una sola fila de casillas, sino de cincuenta. Hemos pasado a una segunda dimensión, a las Variables Indexadas de segundo orden o Series Bidimensionales, —o más corto— a las matrices.

Lanzados como estamos, ¿por qué no ir a una tercera dimensión, o a una cuarta, o a una quinta? Sólo hay dos inconvenientes. Uno es el consumo de memoria interna, a veces terrible, que estas Variables Multidimensionales pueden producir. Y otro es el vértigo que le entra al programador cuando se mete en estos berenjenales.

Vamos con un ejemplo completo. Tenemos un negocio de distribución de chocolate en todo el ámbito nacional y contamos con ocho vendedores. Ha terminado el ejercicio y queremos ordenar el volumen de ventas por vendedor. Escogemos VVV como nombre de nuestro vector. El importe total de las ventas del primer vendedor irá a parar a VVV(1), el del segundo a VVV(2) y así hasta VVV(8).

Preguntamos: aunque todos los elementos del vector tienen el mismo nombre genérico, VVV. ¿Son independientes cada uno de los valores concretos VVV(1), VVV(2), etc., o hay posibles filtraciones entre ellos?

Respondemos: son totalmente independientes por mucho que las tres uves sean idénticas. Nada tiene que ver lo que ha vendido Pedro con lo que ha vendido Cosme. Los compartimentos permanecen estancos.

Pero enseguida vemos que la información es incompleta. Aparecen discrepancias sobre las previsiones. Hay que profundizar más. Necesitamos el detalle de las ventas por provincias. Entonces, con

entera libertad definimos la matriz VVP, de 2 dimensiones, en la que:

VVP(4,1) será el total de las ventas que el vendedor número 4 ha hecho en la provincia 1.

VVP(2,28) será el volumen de ventas que el vendedor número 2 ha hecho en la provincia 28.

Y así todos.

Pero aún no es bastante. Queremos llegar aún más al fondo: saber lo que cada vendedor ha sido capaz de colocar no sólo por provincias sino también por meses. Y, tranquilamente, creamos la matriz tridimensional VPM:

VPM(5,2,7) será el volumen de ventas que el vendedor número 5 ha conseguido hacer en la provincia 2 durante el mes 7, o sea, en julio.

Etc.

Pero nuestras ansias de escudriñar no conocen límite. Y como resulta que hay diez clases de chocolate, exigimos se nos dé cumplida información de todo lo anterior y además, por clases de chocolate. Y para ello, ahí nos espera la matriz tetradimensional VPC para complacer nuestros deseos. En ella:

VPC(8,48,12,10) será la «casilla» (bastante difícil) de representar en cuatro dimensiones) en la que estará la cifra que corresponda a las ventas que el vendedor número 8 ha hecho en la provincia número 48, en el mes 12 del chocolate clase 10.

Hágase cargo el lector que el número de casillas (desde la VPC(1,1,1,1) hasta la VPC(8,48,12,10) que habría en juego, sería de 8*50*12*10 = 48000 (!!). Y ocupando algo más de 4 bytes cada casilla (los vectores producen un pequeño consumo adicional de memoria, en comparación con las variables simples), lo que nos lleva a un total de ocupación cercano a los 200000 bytes (200 Kbytes), que no es grano de anís.

Hacemos observar que aunque seguramente —si lleváramos a la realidad nuestro ejemplo— hubiera una gran parte de las casillas vacías (es muy probable que los vendedores se repartan las provincias y que su actividad comercial alcance a unos pocos meses al año), eso no influiría en el consumo de memoria. He aquí por qué es este motivo, y no el vértigo del programador, la gran limitación del uso de las matrices en la programación.

Con los datos del ejemplo, las ocupaciones alcanzarían en los anteriores supuestos:

- 1 Dimensión: Unos 40 bytes.
- 2 Dimensiones: Unos 2000 bytes.
- 3 Dimensiones: Unos 24000 bytes.
- 4 Dimensiones: Unos 200000 bytes.

Constatamos cómo se disparan las cifras cuando se pasa de una dimensión a la siguiente.

La gran ventaja del uso de las matrices es la facilidad con que el programador puede manejar los datos de ellas. En el ejemplo anterior sería trivial saber;

- El chocolate que se ha vendido en Málaga en todo el año.
- El chocolate que ha vendido Gómez.
- Todo el chocolate que se ha vendido en octubre.
- (En fin, cualquier agrupamiento que se nos ocurra).

OPERADORES

OPERANDO, OPERADOR, OPERACION:

Operando es cualquier número, cadena, expresión numérica o alfanumérica que va a ser sometido al tratamiento de un operador. Operador será pues *lo que* va a actuar sobre el o los operandos y por último, por operación se entiende la acción completa de este tratamiento. Este trabalenguas, difícil de digerir, se va a ver con más claridad en el siguiente ejemplo:

```
IMPORTE = PRECIO * CANTIDAD
```

Las variables PRECIO y CANTIDAD son los operandos, el asterisco (*) es el operador y el conjunto de PRECIO*CANTIDAD (léase precio por cantidad), es la operación, cuyo resultado va a pasar a la variable importe.

OPERADORES ARITMETICOS

Los que actúan sobre operandos numéricos. Son estos:

- Exponenciación: ^ P.ej.: 6^5 (léase «seis elevado a cinco») .

— Multiplicación: * P.ej.: 8.3*5.1 (léase «ocho con tres por cinco con uno»)

— División: / P.ej.: 10/32 (léase: «diez dividido por treinta y dos»)

— Suma: + Omitimos por obvio el ejemplo

— Resta: - Idem de lienzo.

Ahora vienen los exóticos, muy útiles por cierto y que, *atención,* sólo pueden emplearse con variables o constantes *enteras.*

— División entera: \ (Ojo, se trata de la barra invertida) P.ej.: 47\6 (léase «cuarenta y siete dividido entero por seis») cuyo resultado será 7.

1120\10 valdrá 112 y 33\34 valdrá 0.

— Módulo: *MOD* P.ej.: 23 MOD 4 (léase «veintitrés módulo cuatro»). El valor que toma esta expresión es el del *resto* de la división de 23 entre 4, o sea, 3.

20 MOD 5 valdrá 0 y 999 MOD 1000 valdrá 999.

Insistimos: estos dos últimos operadores dan error si se intenta emplearlos con variables reales. Más exactamente, cuando se intenta emplearlos con valores que superen los límites de las variables enteras.

Cuando se utilizan varios operadores aritméticos en una misma expresión, el orden en que funcionan es el expresado, con empate para * / y lo mismo para + - y para \ MOD.

Por ejemplo en la expresión:

22.1+7.4*12/3^9

primero el 3 se elevará a 9, luego el 7.4 se multiplicará por 12 dividiéndose a continuación ese resultado por el de la primera operación, y sumándose finalmente esto a 22.1.

No hay que romperse la cabeza. Todo esto, no es nada nuevo para quien ostente el título de bachiller elemental ya que es lo mismo que sucede en las expresiones algebraicas a las que estamos (o estábamos) acostumbrados.

Y si queremos cambiar el orden de preferencia también haremos lo que en el bachillerato hacíamos: echar mano de los socorridos paréntesis:

1.2*(3.44-1.6)/(6.8+13)^2 Entreténgase el lector en averiguar el resultado

En caso de duda, no vacile Vd. en abusar de los paréntesis.

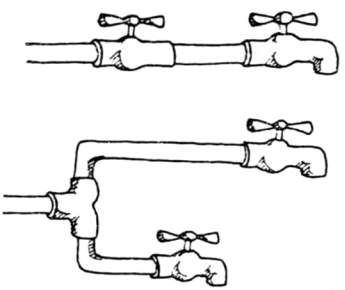
Don Carmelo, en su clase de antaño, probablemente se enfadaría muchísimo si le viese escribir (3/4)-(1/7). Más correcto es, desde luego, poner 3/4-1/7, pero al ordenador le va a dar exactamente lo mismo. Y a Vd. nadie le llamará chapuza (aunque realmente lo sea, eso sí).

OPERADORES RELACIONALES O DE COMPARACION

Son los que comparan *dos* valores numéricos o alfanuméricos. Son seis:

- Igualdad: = P.ej.: A=B (léase «A igual a B»)
- Superioridad: > P.ej.: A>B («A mayor que B o A mayor de B»)
- Inferioridad: < P.ej.: A<B («A menor que B o A menor de B»)
- Desigualdad: < > P.ej.: A<>B («A distinto de B»)
- Mayor o igual: > = P.ej.: A>=B («A mayor o igual que B»)
- Menor o igual: < = P.ej.: A<=B («A menor o igual que B»)

OPERADORES LOGICOS



Nos limitaremos ahora sólo a enunciarlos:

- NOT (No)
- AND (Y)
- OR (O)
- XOR (O Exclusivo)
- IMP (Implicación)
- EQV (Equivalente)

De estos seis operadores, el AND y el OR se emplean con mayor frecuencia que el resto.

A bombo y platillo queremos anunciar hic et nunc que estos operadores, junto con los de comparación, se llevan la mayor parte del peso de la razón de ser de los ordenadores. Sin ellos, nos habríamos quedado, todo lo más, en la gran calculadora.

Tendremos oportunidad en el desarrollo del Basic de ampliar estas nociones y de practicar a fondo tanto con estos operadores lógicos como con los relacionales, vinculados especialmente a la sentencia IF.

OTRAS CONSIDERACIONES DE INTERES

Para ya entrar en materia plena, queremos dejar tomada nota de unos cuantos consejos, advertencias y simplificaciones que creemos fundamentales para llevar a cabo un eficaz trabajo de programación con el Basic de Microsoft.

COMPILACION E INTERPRETACION

La diferencia entre estas dos formas en que puede ser ejecutado un programa es algo que raras veces se entiende por completo. Y es necesario llegar a su total comprensión.

Para que un programa sea ejecutable en intérprete es necesario que en memoria interna estén presentes (aunque separados) dos programas:

- Nuestro propio programa, el que queremos ejecutar. Se le denomina «programa-fuente».
- El programa-intérprete capaz de traducir al lenguaje-máquina las instrucciones que hemos introducido en nuestro programa-fuente.

Cuando se ejecuta un programa en estas condiciones, *todas* las instrucciones que lo componen —en lenguaje de alto nivel, en Basic en nuestro caso—, y ordenadamente, *una por una* pasan, quieran o no, por el traductor, por el programa intérprete, quien se encarga de llevar a cabo la traducción de cada instrucción para que el microprocesador entienda lo que queremos decir. Esta traducción tiene lugar con *cada una* de las instrucciones al modo de las conocidas «traducciones simultáneas» radiadas o televisadas.

Si, pongamos por caso, el programa tiene que ejecutar un bucle de 5000 vueltas, el intérprete tendrá que efectuar 5000 veces la traducción de cada una de las instrucciones que componen el lazo, sin clemencia para nadie.

A pesar de las increibles velocidades con que el ordenador trabaja, el procesado de un programa en intérprete da a veces sensación de lentitud debido al cúmulo de operaciones que la interpretación lleva consigo. Y en ocasiones el programa se ejecuta con desesperante premiosidad: ya no se trata de una simple *sensación* (es asombroso cómo se hacen de largos los segundos cuando esperamos que aparezca un dato en la pantalla).

En un programa *compilado* en cambio, todas las instrucciones que lo componen, han sido previamente *traducidas* a lenguaje-máquina. Ya no hace falta el programa-intérprete en memoria interna porque ya no hay nada que traducir: todo él lo está de antemano. En memoria interna sólo estará presente esa traducción global en la que si metemos la nariz (hay modos de hacerlo) muy difícilmente entenderemos algo. En cambio para la máquina las cosas se facilitan sobremanera. (No tiene que traducir una a una las palabras de Mister Reagan porque le han sido entregados unos folios con la traducción *completa* ya hecha).

El programa así traducido recibe el nombre de «programa-objeto».

Al comparar los programas interpretados y compilados la única gran ventaja de los primeros estriba en su facilidad de depuración. Un programa escrito en nuestro Basic puede ser probado y corregido casi de línea en línea. No debe olvidarse a este respecto que el tiempo de depuración de un programa suele superar con creces al de la propia programación.

Pero cuando un programa ya está suficientemente probado *(quemado* suele decirse), si a continuación puede compilarse, ello no comporta más que ventajas.

Un programa compilado *rueda* a una velocidad de 1.5 a 30 veces superior al interpretado. Además, el programa no puede *tocarse*, no corremos el riesgo de que el usuario o el operador pueda meter mano en él, con grave riesgo para su funcionamiento, de acuerdo con la implacable ley de Murphy («si hay porquería en un programa, ésta, más pronto o más tarde, saldrá»; en inglés «garbage in garbage out»).

Por otra parte el trabajo de compilación (no por parte del ordenador, claro, sino del programador-operador) es extremadamente sencillo y no entraña riesgo alguno.

Pues bien, *el Basic de Microsoft reúne las ventajas de los lenguajes interpretables y las de los compilables.*

Además de los dos grandes ventajas mencionadas a favor de la compilación, citemos también otra cualidad no menos ventajosa.

Es muy recomendable (y desgraciadamente se hace poco y mal) documentar los programas tomando provecho de la sentencia REM (véase). Estas líneas aclaratorias intercaladas en el programa consumen gran cantidad de memoria mientras se permanece en fuente (en intérprete). Al compilar, en cambio, estas líneas son desechadas inmediatamente por el compilador, es decir, no consumen ni un solo byte de memoria interna.

Asimismo, cuando se trabaja en intérprete, es corriente procurar que los nombres de las variables sean cortas por el mismo motivo. Si hemos de utilizar una variable numérica que represente las existencias de un artículo en un almacen, la denominaremos, por ejemplo *EX* que gasta menos memoria que si la llamásemos *EXISTENCIAS*. Bien, pues resulta que al compilar, es indiferente el número de caracteres empleados en la denominación de la variable porque —tremenda astucia— el compilador hace caso omiso de esa denominación ya que lo único que retiene es la *dirección física* donde se encuentra el valor de esa variable.

La única servidumbre relativa que presenta el compilador es que exige algo más de disciplina a la hora de programar. Y que proscribe el uso de algunas sentencias. Pero como esas restricciones son menores y de ninguna manera suponen una verdadera limitación ni pérdida de libertad en nuestra programación (en fuente, en Basic), en todo el resto de este texto iremo dejando constancia de todas esas restricciones. De este modo el lector estará en disposición de llegar hasta la última fase de aprendizaje del Basic.

NUMERACION DE LINEAS DE PROGRAMA

Los números con los que se inician cada una de las líneas que componen un programa, puede tomar cualquier valor entre 0 y 65529, ambos inclusive. Puesto que un programa, por largo que sea nunca hará uso ni de la vigésima parte de los números de línea que en teoría podría utilizar, se recomienda vivamente que se adopte un intervalo de numeración no inferior a 10. Quizá 20 sea preferible. Esto permite la intercalación posterior de líneas con instrucciones que sin querer hemos olvidado, mejoras de programación, etc.

Para animar a que así se haga, recalcamos que el Basic reordena automáticamente la numeración del programa en orden creciente. Podemos, por ejemplo, saltar de la línea 400 a la 1000 porque

deseemos dejar ahí un vacío para posterior uso.

En este Basic (no en todos) la longitud de una lína de programación puede alcanzar 255 caracteres (254 si pensamos compilar después).

Haciendo cuentas, y suponiendo que se trabaja con pantalla de 80 caracteres por línea, parece entonces que sólo se podrán llenar tres líneas físicas y pico con una línea de programa. En realidad se pueden utilizar muchas más ya que, como más tarde veremos, uno de los caracteres ASCII importantes (el 10) es el *Salto de Línea* (LF, Line-feed). Este carácter puede incorporarse impunemente en nuestras líneas de programa sin más que dar CONTROL-J, con lo que automáticamente veremos al cursor saltar una línea más abajo. Cada uno de estos saltos forzados representa el consumo de *un sólo* carácter a descontar de los 254.

Cada línea puede tener tantas instrucciones como la limitación anterior —y la propia lógica del programa, claro— lo permita.

Para terminar una línea de programa, más exactamente, para hacer saber al Basic que una línea de programa está completa, que no vamos a añadir ninguna instrucción más, basta con pulsar la tecla CR. En ese momento esa línea se introduce en el lugar que le corresponda de acuerdo con su numeración.

Aun a riesgo de parecer machachones, volvemos a insistir en que cuando decimos «una línea de programa» es posible que esa «línea» contenga una, cuatro o diecisiete líneas físicas, renglones de pantalla. Y que no hay diferencia entre «línea de programa» y «línea de instrucciones». Por lo que una línea así entendida tambien puede contener una, siete o veintidós instrucciones. Prometemos no volver a repetirlo.

Cuando se está ejecutando un programa y éste pasa por una línea que contiene un error de sintaxis (todo lo que se introduce en un programa que sea ininteligible para el intérprete Basic lo califica éste de *Sintax error*) el programa se interrumpe, apareciendo en pantalla la línea en la que existe el error y dejando preparado el cursor para su inmediata corrección.

CORRECCION DE UN PROGRAMA

Retomamos la última línea del párrafo anterior. Cuando el cursor está en esas condiciones, el Basic nos sitúa de una manera automática en el Modo Edit o Modo Corrección que anunciábamos en el capítulo anterior.

Bien, ante todo hay que decir que si (estando en Modo Control) escribimos un número de línea que exista en el programa y damos CR, acabamos de *matar* a esa pobre línea. Desaparece del programa de una manera irrecuperable. Esa es una manera de eliminar una línea. No es la única.

Si escribimos el número de esa línea y las instrucciones pertinentes con un CR al final, la nueva línea desplaza a la que hubiera, borrando todo vestigio de lo que anteriormente contuviese.

Esa es una manera obvia de corregir. Pero el exigente lector tiene derecho a sentirse decepcionado por la rudeza de los procedimientos descritos. En efecto, existen otros más elegantes y menos laboriosos.

Y nuevamente volvemos al primer párrafo de este apartado. Habíamos cometido el error de sintaxis y lo queramos o no se nos había situado en Modo Edit. Muy bien. También podríamos haber llegado a él de forma voluntaria (desde modo control) escribiendo:

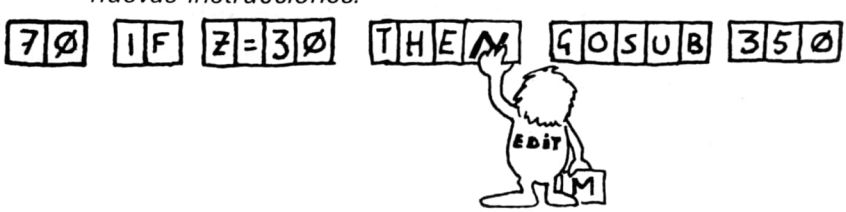
EDIT 30 ⟨CR⟩

Esto nos llevaría exactamente a la mismísima situación que a la que nos llevó, sin nuestra intervención, el *Sintax error.*

(Si escribiésemos *EDIT.* nos aparecería, lista para corrección, la *última* línea introducida o corregida o listada).

Vale: ¿y ahora qué? Podemos hacer muchas cosas, a saber:

* Pulsar CR, con lo que la línea, tal y como estaba, vuelve a su sitio. Todo queda como al principio.
* Avanzar sobre la línea, simplemente manteniendo apretada la barra de espacios (con lo que se va reproduciendo la línea en cuestión). O ir hacia atrás apretando la tecla de retroceso o Control H, con lo que se reproduce la línea pero marcha atrás.
* Pulsar la tecla L con lo que nos *lista (aparece en pantalla) la línea en cuestión, sin que se nos saque del Modo Edit.*
* *Pulsar la tecla X, que nos reproduce la línea dejando el cursor al final de la misma dispuesto a que podamos agregar nuevas instrucciones.*



* *Pulsar la tecla Q (de Quit* = abandonar) con lo que nos escapamos del Modo Edit y recuperamos el modo control. Como cuando pulsamos CR, tampoco ha pasado nada. Hay una diferencia entre estos dos casos, no por sutil menos importante. El uso de la Q deja sin tocar el contenido de la memoria. El valor que pueda tener cualquier variable permanece intocado. No sucede lo mismo con el CR que implica que se ha producdo una *edición* y esto (la modificación del programa, por mínima que sea) acarrea una anulación (inicialización) de todas las variables.
* Pulsar la tecla S (de *Search* = buscar). En apariencia no ha pasado nada. Pero en la realidad el ordenador está esperando a que Vd. apriete otra techa. Y será el carácter pulsado el que *buscará* desplazándose el cursor instantáneamente hasta él. (Si no encuentra ninguno irá al final, con evidente sentimiento de fracaso).
* Pulsar D (de *Delete* = borrar. Cuidado). Queda borrado el carácter que se halle bajo el cursor. Para borrar varios caracteres contiguos se escribirá el número de caracteres a eliminar y luego una D. O pulsar la D tantas veces como caracteres quieran borrarse.
* Pulsar K (de *Kill* = matar. Más cuidado). Borrado de *todos* los caracteres que haya desde el principio de línea hasta el último carácter pulsado.
* Pulsar H (Muchísimo cuidado). Elimina todo lo que haya desde la posición del cursor hasta el final de la línea.
* Pulsar I (Insertar). Lo que a continuación se escriba queda intercalado en el lugar en que se haya hecho la inserción. Para salir de ella hay que apretar la tecla de escape (ESC en unos ordenadores y BREAK en otros).
* Pulsar C (Cambiar). Sustituye el carácter que se halle bajo el cursor por el que se pulse a continuación. Para sustituir varios caraceres contiguos se escribirá el número de caracteres a cambiar seguido de una C y luego se escribirán los caracteres nuevos.

Todo lo anterior puede que produzca desaliento en el novicio en estas lides. Nada más cómodo y sencillo en la práctica. Es como tantas otras cosas de informática (y de la vida en general) en las que la explicación de algo sencillo no puede escapar a la apariencia de farragoso. (¿Ha pensado alguna vez cómo describiría Vd. a un habitante inteligente de otra galaxia lo que es la *Izquierda* —física, no política— pongamos por caso?)

Veamos un ejemplo (no importa que no se entienda lo que significan las instrucciones. Se trata de practicar el EDIT).

Acabamos de introducir la línea 320, que exactamente es ésta:

```
320 FOR I=1 TO 1000:Y$=INKEY$:IF Y$=CHR$(2) THEN PRINT
    "SALVESE QUIEN PUEDA" ESLE IF Y$="F" THEN 440
```

Hemos cometido varios errores:

- Hemos puesto 1000 en lugar de 100
- En vez de CHR$(2) deberíamos haber escrito CHR$(11)
- Nos hemos confundido escribiendo ESLE donde queríamos poner ELSE.
- Nos ha faltado añadir al final «ELSE END».

Paso a paso. En primer lugar escribiremos *EDIT 320* o mejor, aún, *EDIT.* En cualquier caso, además, CR.

Aparecerá en la pantalla el número 320, un espacio en blanco y el cursor. Está esperando a que actuemos. Vamos allá.

Apretamos la barra de espacios y veremos brotar letras.

```
FOR I=1 TO 1..
```

Cuando el cursor se encuentre sobre cualquiera de los tres ceros de 1000 soltamos la barra espaciadora. Apretamos la D con lo que veremos que el 0 que estaba debajo del cursor queda emparedado entre dos barras. Eso significa que ese cero está eliminado.

Ahora apretamos la L y nos aparece la línea completa. Y otra vez la L para que comprobemos que el 1000 se ha convertido en 100. Correcto. La situación es análoga a la inicial, con el cursor a comienzo de línea.

Volvemos a apretar la barra de espacios hasta ponerlos encima del *2* de CHR$(2). Hay que cambiarlo por *11.* Ay, nos hemos pasado. El cursor está encima de la *T* del *THEN*. Retrocedamos hasta el 2 para lo cual manteniendo apretada la techa de control pulsamos tres veces la H (3 retrocesos).

Ya está el cursor encima del *2.* Apretamos la C y el 1. Hemos cambiado así el *2 por un 1. (Hemos conseguido cambiar CHR$(2) por CHR$(1), pero lo que necesitamos en CHR$(11).)*

Pulsemos la I y un 1, con lo que habremos insertado un 1 junto al anterior. Ya tenemos el 11.

Escapamos del modo Insert para lo que pulsamos ESC (o BREAK).

Volvemos a apretar la barra de espacios para avanzar hasta el ESLE que hemos de convertir en ELSE. Dejamos que el cursor se pose sobre la S del ESLE erróneo. Ya está.

Ahora podemos actuar de muchas formas. La más fácil parece ser la de escribir CLCS. Otra manera: 2CLS. Y otra: DDILS. (Invitamos al lector a que le siga la pista a estos tres modos de cambiar ESLE por ELSE y aún, que trate por sí mismo de encontrar algún otro procedimiento de enmendar el error.)

Por último, tenemos que incluir al final *ELSE END*. Basta con pulsar X y escribir ese apéndice.

Algún avisado lector nos dirá que seguramente se hubiese tardado menos en repetir la linea de nuevas. Bueno, en este caso, con cuatro errores, desde luego que sí. Nos remitimos a lo que antes hemos comentado: todas las operaciones descritas son, en la práctica muy rápidas. Y también es cierto que la costumbre lo hace casi todo.

La pulsación de CONTROL A produce automáticamente el "Modo EDIT" con respecto a lo último que hayamos escrito. Si eso último ha sido una línea de programa —atención— lo que queda sujeto a *edición* (modificación) es el *contenido* de la línea, es decir, sin incluir el número de la misma.

Veamos una aplicación muy útil de esta función de control.

Supongamos que acabamos de teclear una larga línea de instrucciones con su CR final. Y de pronto nos percatamos de que el número de línea no es el correcto. Nos resistimos a tener que repetir la larga sarta de instrucciones. Operaremos de la siguiente guisa:

Pulsemos CONTROL A, con lo que aparecerá un ! y el cursor a su derecha. Ya estamos en nuestro modo EDIT. Si apretamos la *L,* tal como acabamos de ver, surgirá en pantalla la línea que terminamos de introducir *pero sin número de línea.* Muy bien. Para que esas instrucciones vengan precedidas del número 1220, que es el deseado, no tendremos más que pulsar una *I* (insertar) y teclear *1220* y CR con lo que acabamos de incorporar al programa la línea 1220 con todo lo demás.

CARACTERES DE CONTROL

Resumimos los efectos producidos por la pulsación de los caracteres de control:

CONTROL-A: Sitúa en modo EDIT a lo último que se haya introducido por teclado.

CONTROL-C: Interrumpe la ejecución del programa que se esté rodando.

CONTROL-G: Suena el *beep* o pitido característico de la máquina.

CONTROL-H: Suprime el carácter que esté a la izquierda del cursor.

CONTROL-I: El mismo efecto que la tecla "TAB" (tabulación).

CONTROL-J: Carácter de salto de línea (line-feed o LF).

CONTROL-M: Exactamente el mismo efecto que pulsar CR.

CONTROL-O: Sin que deje de funcionar el programa, suspende toda *salida* por pantalla o impresora. Un subsiguiente CONTROL-O reanuda la salida interrumpida.

CONTROL-R: Reproduce la línea que se esté tecleando.

CONTROL-S: Suspende —sin abortarlo— la ejecución del programa en curso.

CONTROL-Q: Reanuda la ejecución de un programa suspendido por un CONTROL-S. En algunos sistemas, el CONTROL-S desempeña las dos funciones al modo de un interruptor eléctrico.

CONTROL-U: Elimina la línea que se esta tecleando.

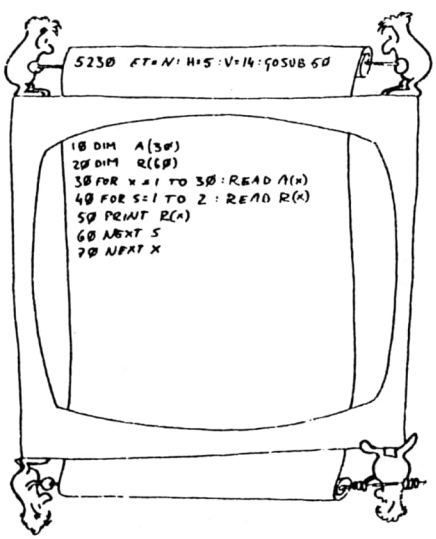
Aunque implícito en lo anterior, recalcamos que las funciones de control *C O S y Q* son operativos *durante* la ejecución del programa. El resto se utilizan en modo control Basic.

CAPITULO 3.— PRIMEROS COMANDOS Y SENTENCIAS. ASIGNACIONES

Conozcamos para empezar y por encima, el significado de media docena de comandos —los fundamentales del Basic: LIST, RUN, LOAD, SAVE, NEW y CLEAR— y de una sentecia (PRINT), indispensables para ponernos en marcha.

Este lote de directivas forman parte de *todos* los Basic, incluido, claro, el de Microsoft. Y, dentro de este último, todo lo que se teclee desde el BASIC, a excepción de las constantes alfanuméricas, podemos escribirlo con mayúsculas o con minúsculas: el propio intérprete se encargará de "mayusculizarlo" todo.

(Comoquiera que el acostumbrarse a "programar en mayúsculas" comporta ventajas no desdeñables como más adelante veremos, recomendamos al lector así lo haga. Por favor, sin traumas).



LIST: Al escribir estas cuatro letras, precedidas de un «CR» (para salirnos de una línea de programa en la que pudiéramos encontrarnos) y seguido de otro, nos *lista* (saca en pantalla) el contenido del programa que hayamos introducido, tal cual lo hayamos hecho (aunque con la líneas ordenadas de menor a mayor aún cuando las hayamos introducido en cualquier orden), completo y con sus errores si los tuviere.

Si el programa es más largo que lo que cabe en la pantalla, se producirá un "scroll", es decir, irán desapareciendo por arriba las primeras líneas y apareciendo por abajo las últimas.

La digitación de CONTROL-S, como ya hemos visto en el capítulo anterior, detiene inmediatamente este scroll. Y CONTROL-Q lo reanuda.

Si queremos interrumpir del todo el listado, daremos CONTROL-C.

Es este el momento oportuno de decir algo respecto a la forma en que una línea de programa puede ser "redactado". Al contrario que en otros Basic, en el de Microsoft, *los espacios* entre sentencias, variables, argumentos, etc., son *necesarios*. Y nos apresuramos a añadir que, en cambio, los espacios *extra* que el programador guste agregar, no son tenidos en cuenta por el Basic. Es decir, que puede Vd. utilizar la tecla de espacios o la de tabulación tanto como se le antoje. Eso no va a influir en el desarrollo del programa. Sólo va a producir un consumo de memoria, que desaparece al compilar.

RUN: (=*correr, rodar, ejecutar* en imperativo) también precedido y seguido de un CR, es el que ordena hacer funcionar el programa que hayamos introducido. Dicho de otro modo, el ordenador busca la primera línea de instrucciones, y una a una y por su orden, las ejecuta hasta el final con una encomiable obediencia y sumisión.

El respeto del Basic y del ordenador por las instrucciones, junto con su total falta de imaginación para enmendar la plana al programador, puede hacer que *salte* un error, si es que lo hay, interrumpiéndose la ejecución del programa.

Pero puede ocurrir que exista un error para el hombre que no lo sea para el Basic: por ejemplo que hayamos permitido que exista el día 32 de Juliembre. En este caso no ha-

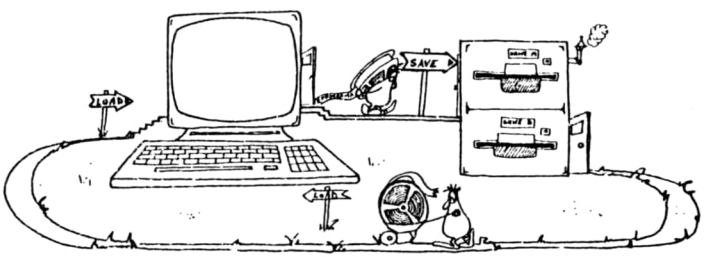
brá interrupción del programa aunque sí, seguramente, sorpresas posteriores poco agradables.

El comando anterior produce su efecto sobre un programa que *ya esté presente* en la memoria interna. Si lo que queremos es ejecutar un programa que no lo esté pero que sí exista grabado en la externa, haremos que se lleve a cabo la ejecución del mismo, agregando al comando RUN el nombre del programa entre comillas.

Por ejemplo:
```
RUN "NOMINA1"
RUN "B:ESTADIS"
```

En el primer caso, el programa buscado en el disco o diskette por defecto, mientras que en el segundo caso, el programa será cargado del disco B.

LOAD: En los dos ejemplos anteriores, la ejecución del programa se lleva a cabo *previa* carga del mismo en memoria interna. Esta transferencia se efectúa sin menoscabo de la integridad del programa grabado, que continuará en su soporte magnético. En memoria interna hay una copia del mismo que es la que se ejecuta. Pues bien, el comando LOAD, (= *cargar*) seguido del nombre del programa, es el que hace esa transferencia, sin más. Sin ejecutar ni una sola línea del programa, éste pasa a memoria interna y en ella se queda para lo que Vd. guste mandar.



SAVE: *(= salvar).* Es el reverso del anterior. Es decir, cuando lanzamos por el teclado este comando, seguido del nombre del programa con el que queremos que quede grabado en la memoria extena, aquél pasa allí tal cual esté en la interna. Dada la volatilidad de la memoria interna, de la que todo desaparece con un simple corte de fluido eléctrico, un programador que se precie, debe usar y abusar de este comando si no quiere llevarse algún día un serio disgusto.

Seamos fiel al lema:

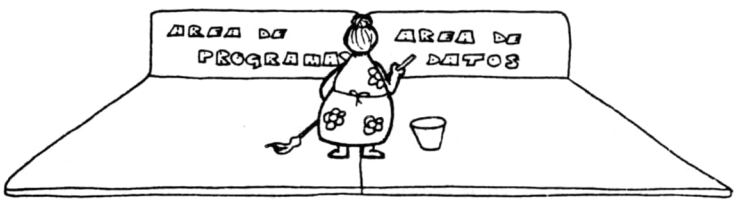
"EL QUE SABE, SAVE" (= "El que sabe, salva")

O al equivalente, cambiando causa y efecto:

"EL QUE SAVE, SABE" (= "El que salva, sabe")

Que así sea.

NEW: *(=nuevo).* Con estas tremendas, aparentemente inofensivas y peligrosas tres letras dejamos la memoria interna (el área de usuario: programación y variables) limpia como una patena; hacemos desaparecer en un instante el trabajo, tal vez de horas. Utilícese pues, este comando con exquisita prudencia y sólo cuando estemos bien seguros de que todo lo que había que salvar está salvado.



CLEAR: (=clarear, *limpiar).* Semejante al anterior, aunque mucho menos peligroso. Este comando se limita a dejar libre, a limpiar, solamente el área de datos, dejando intacto el programa que en memoria interna pudiere existir.

PRINT: Con esta sentencia se ordena escribir, *pintar,* "hacer aparecer en la pantalla" ("displayar" le dicen algunos empedernidos informáticos dispuestos a todo) lo que vaya a continuar del PRINT. Es una sentencia que se emplea tanto que, cuando se está programando desde el intérprete, se puede sustituir sus cinco letras por un simple signo de interrogación (?). El BASIC se encarga de traducir este signo por el PRINT.

Si tecleamos, por ejemplo:

```
PRINT ANTONIO (o lo que es lo mismo: ? ANTONIO)
```

es muy probable que nos llevemos la primera decepción de nuestra brillante carrera de programadores.

Porque en la pantalla no vemos ningún "ANTONIO" sino un cero mondo y lirondo. Claro. Recordemos el capítulo de "variables númericas". El Basic ha entendido, y con toda la razón, que buscabamos el valor de la variable numérica llamada ANTONIO. Que no vale mas que cero. Como todas, si no se le ha asignado con anterioridad algún valor.

Probemos ahora a teclear:

```
PRINT "ANTONIO" (o  ?"ANTONIO")
```

Ahí está. Sin bigote, pero sin que le falte una letra.

Y ahora:

```
PRINT 4444
```

Sale el número. Ni uno más ni uno menos. Pero aparece un espacio antes del 4444. Es para el signo. Que si es positivo (+), como ocurre en nuestro caso, no lo pone. (Es el signo por defecto).

En cambio:

```
PRINT -1000
```

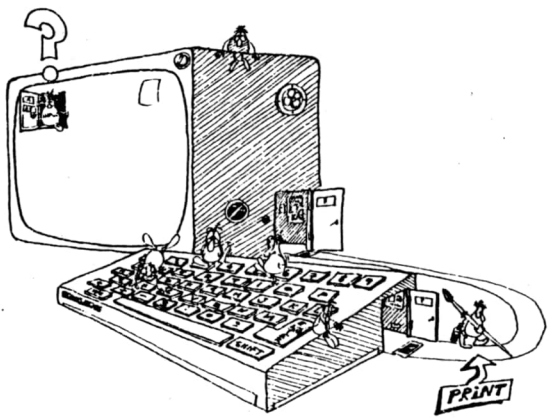
Aparece el -1000 sin espacio previo. Por ahora ya basta. Volveremos sobre el PRINT, que tiene mucho que decirnos.

En caricatura y en cualquier lenguaje, un programa no va a ser sino el juego, más o menos sofisticado, de un conjunto de datos.

Y los datos sólo pueden adoptar dos formas: Numérica y Alfanumérica. Unos y otros se manejan mediante el empleo de variables o de constantes.

A los datos alfanuméricos (ya lo sabemos) se les denomina "cadenas". Porque lo son. Cada eslabón es un carácter ASCII (una letra, un guarismo, un carácter especial como el « # » o incluso un carácter no visible como el retroceso). Cualquier "cosa" que se corresponda con un número comprendido entre el 0 y el 255 puede

formar parte de una cadena alfanumérica. Más adelante volveremos sobre la "cosa".



ASIGNACION DE VARIABLES

Para que una variable de cualquier tipo tenga un valor, es preciso que se lo hayamos dado. Nueva perogrullada. Si no, su valor será 0, valor por defecto de todas las variables. Machacona pero necesaria insistencia la nuestra.

Hay varias maneras de asignar (dar) valores a una variable.

1.— ASIGNACION DIRECTA

Es la genuina. Así de simple:

```
X=43      o      M$="Mangulete"      o      M(2)=28
```

Las variables X y M$ "valdrán" 43 y "Mangulete" respectivamente. Y el segundo elemento del vector M valdrá 28. Hasta que otra asignación los cambie, con ellos permanecerán.

En los Basic primerizos esta asignación directa debía hacerse con la sentecia LET delante de la variable.

```
LET X=43      o      LET M$="Mangulete"      o      LET M(2)=23
```

(Al castellano lo traduciriamos por "SEA X=43, etc.).

En el evolucionado Basic de Microsoft, podemos seguir utilizando el LET. Pero como no es necesario, nosotros no lo vamos a hacer.

Queda claro, pues, que la asignación de un valor a una variable es:

Nombre de la Variable = Valor asignado

Parece que no hay nada mas que comentar. Pues si. Hay que aclarar algo. Y con solemnidad.

En las expresiones anteriores estamos haciendo uso de un signo que nos es muy familiar: el de igualdad (''='). Pues no nos hagamos ilusiones. A pesar de su apareciencia *no se trata de una igualdad* en sentido estricto.

Si lo fuese, podríamos poner 43=X con el mismo derecho que X=43.

Con ello cometeríamos una gran torpeza. Si de veras lo hacemos, hemos introducido una extraña instrucción en la línea 43:

```
43 = X
```

Que no sirve mas que para que, al ''darle RUN'' nos digan:

Syntax error in 43

Si no podemos cambiar los miembros de la igualdad es por una razón obvia: no es una igualdad, es una asignación, que no es lo mismo. Si se utiliza el signo ''='' es debido a una cuestión de pura economía informática.

Nosotros, desde ahora y para siempre, cuando escribamos o leamos ''X=43'' deberemos traducirlo mentalmente por *''X, te ordeno que a partir de ahora valgas 43''.* Y esto es universal en Basic.

Otro ejemplo. La instrucción:

```
X1=(-B+SQR(B^2-4*A*C))/(2*A)
```

que, a lo mejor a alguien le recuerda a la fórmula para la obtención de la primera raiz de una ecuación de segundo grado, y acierta, porque lo es (en formato Basic), esta asignación, decimos, hace que X1 valga el resultdo de la expresión que figura a la derecha del signo de igualdad.

Digamos pues, brevemente y con la solemnidad anunciada, que *en toda asignación, la izquierda RECIBE y adquiere el valor de lo que encuentra a la derecha.*

(Atención: El «RECIBE» no significa que le suceda nada a lo que haya a la derecha. Si ponemos A = B, automáticamente A pasa a tomar el valor que pueda tener B, pero B no ha perdido lo que tenía; sigue con el mismo valor que tuviera antes de la asignación.

Con dudosa generosidad B cede, sin perder, su contenido a A. Se produce, por consiguiente, una simple copia de derecha a izquierda).

Para terminar con las asignaciones, es menester todavía dejar bien claro que a la izquierda del signo « = » (único signo admisible y siempre que se trate de asignaciones) no puede haber más que una variable, numérica o alfanumérica, pero una sola variable.

No va a permitir el Basic (ni el de Microsoft, ni ninguno) hacer las siguientes barbarasignaciones:

```
Z>X+2   ni   T+1=H/3   ni siquiera   A<=3
```

En cambio, sí que nos dejará incluir en nuestro programa las siguientes asignaciones, en apariencia incongruentes:

```
Z(4)=Z(2)+Z(3)   o   A=A+B   o   N=N+1   y hasta   H=H/1000
```

Veamos lo que pasa en los cuatro casos:

En el primero, el elemento del vector Z que hace el número 4 toma el valor de lo que tengan en ese momento el segundo más el tercero. Supongamos que justo antes de llegar a esa asignación, Z(2) vale 15, Z(3) vale 9 y Z(4) vale 777.

¿Cuánto valdrá cada uno a la salida de la asignación?

(Intente el lector averiguarlo)

La respuesta correcta es 15,, 9 y 24 respectivamente. Del 777 que tenía Z(4) no quedan ni trazas.

Veamos qué ocurre en el segundo caso (A = A + B) suponiendo que los valores previos fuesen 32.6 para A y 600 para B.

A cámara lenta: A recibe el valor de A más el valor de B. Luego recibe 32.6 más 600. Pasa a valer entonces 632.6.

El N = N + 1 del tercer caso, (con N o con la variable numérica que a Vd. se le antoje) constituye una de las expresiones que más abundan en un programa Basic.

Supongamos que nuestra asignación está al principio del programa. ¿Cuánto vale así, de nuevas? Cero, como todas las variables numéricas. Bien.

Ahora se presenta el duro trago de que N pase a valer N + 1. No hay que alarmarse. Hay que sumar 1 a *lo que valía* N. ¿Tan difícil es efectuar la suma de O + 1?

Es natural que quien lo ve por vez primera le escandalicen tamañas contradicciones algebraicas. Hagamos desaparecer el escán-

dalo insistiendo, por última vez, que no se trata de igualdades sino de asignaciones, amén.

Y ya, por fin, podemos confeccionar nuestro primer programa.

Recomendamos vivamente al lector que intente hacer él de ordenador, y anticipar el resultado que éste debe producir.

PR-1.1: Asignaciones sencillas.

```
10 A=13
20 B=176935
30 C=A+B
35 D=-14
40 PRINT A
50 PRINT B
60 PRINT C
70 PRINT D

RUN <CR>
 13
 176935
 176948
-14
Ok
```

No hemos conseguido gran cosa. Pero vamos a complicarnos la vida.

PR-1.2: Más asignaciones.

```
20 AMIGO=123: AMIGO$="IGNACIO"
40 AMIGA=9: AMIGA$="INES"
60 AMIGA=AMIGO+4: AMIGO=AMIGA+6
80 PRINT AMIGO$: PRINT AMIGA$
100 PRINT AMIGO: PRINT AMIGA
110 AMIGO=AMIGO+10: AMIGA=AMIGA+20
130 PRINT AMIGO: PRINT AMIGA

RUN <CR>
IGNACIO
INES
 133
 127
 143
 147
Ok
```

PR-1.3: Asignaciones difíciles de digerir. Pero válidas.

```
10 X=7: PRINT X
20 X=X+3: PRINT X
30 X=X+X: PRINT X
40 X=2*X: PRINT X
50 X=X*X-15*X: PRINT X
60 Y=X/10
70 X=15: X=X^2: PRINT X
80 X=X/Y: PRINT X
90 X=Y^.5: PRINT X
100 X=X-6
110 X=X^X: PRINT X
120 X=Y^1/2: PRINT X
130 X=Y^(1/2): PRINT X

RUN <CR>
 7
 10
 20
 40
 1000
 225
 2.25
 10
 256
 50
 10
Ok
```

La pobre variable X no se cansará de recibir valores. Y los PRINT actuarán instantáneamente. Recomendamos al lector novato que con paciencia, siga la pista a la variable X en sus vicisitudes dentro del programa.

PR-1.4: Resolución de «una» ecuación de segundo grado:

```
40 A=1: B=-3: C=2
60 D=(B^2-4*A*C)^(1/2)
80 X1=(-B+D)/(2*A)
90 X2=(-B-D)/(2*A)
94 PRINT X1: PRINT X2

RUN <CR>
 2
 1
Ok
```

PR-1.5: Asignaciones a variables alfanuméricas.

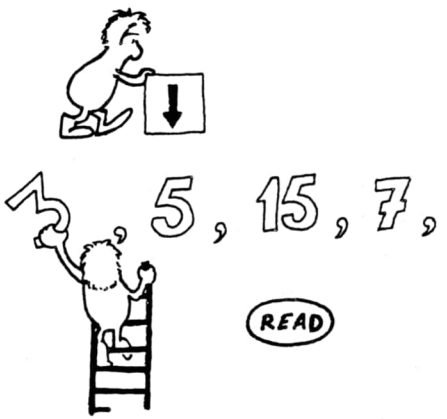
```
20 D$="Beatriz":A$="PATATAS BLANCAS PARA LA CARAVANA"
30 C$="BASIC DE MICROSOFT"
40 M$="&&&#####@@@"
50 PRINT A$: PRINT B$: PRINT C$: PRINT D$: PRINT M$
60 Z$=C$: C$=D$: D$=Z$
80 PRINT C$: PRINT D$
90 A$=M$: M$=A$: PRINT A$: PRINT M$

RUN <CR>
PATATAS BLANCAS PARA LA CARAVANA

BASIC DE MICROSOFT
Beatriz
&&&#####@@@
Beatriz
BASIC DE MICROSOFT
&&&#####@@@
&&&#####@@@
Ok
```

Algo que observar. Hemos pedido que se nos muestre B$, variable a la que no se le había asignado nada. Se ha producido un simple salto de línea. Recuérdese. Y obsérvese también que la línea

60 ha producido un trueque de valores entre C$ y D$, cosa que no ha hecho la línea 90 con A$ y M$, sino más bien una tontería.

2.— ASIGNACION A TRAVES DE LAS SENTENCIAS «READ» Y «DATA»

Con estas sentencias se produce una especie de «asignación en masa». Se trata de uno de los pocos casos en que dos sentencias —que pueden estar muy separadas en el programa— carecen de sentido por sí solas. El «READ» sin «DATA» daría error. El «DATA» sin «READ» estaría de sobra en el programa, pero no produciría error. Julieta (DATA) puede vivir (aunque inútilmente) sin Romedo (READ). Pero Romeo sin Julieta no puede vivir siquiera.



DATA significa «datos», y detrás de esa palabra se escribe eso, datos. Cualesquiera. Separados por comas, que indican la frontera entre ellos. Importante: esos datos sólo pueden aparecer en forma de *constantes* numéricas o alfanuméricas. Nunca en forma de variables.

Las sentencias DATA, gozan del privilegio de poder situarse en cualquier parte del programa. Es indiferente el número de línea donde estén. Son datos que se incorporan en el programa para que la sentencia READ los *lea* (READ en inglés = Leer).

Cuando el programa se encuentra con un DATA, se lo salta olímpicamente, como si no existiese. Cuando es un READ con lo que el programa se tropieza es cuando el DATA cobra sentido. Pero es mejor ver todo esto con un ejemplo.

PR-1.6: Primer intento de READ-DATA.

```
10 DATA 2,3,5
20 READ A: READ B: READ C
30 PRINT A: PRINT B: PRINT C

RUN <CR>
 2
 3
 5
Ok
```

Podríamos anular la línea 10 y escribir la 2000 (por ejemplo) con el contenido de la 10: daría el mismo resultado.

Fijémonos en la línea 20. La primera instrucción (READ A) se traduce por «Lee el primer dato que encuentres y dáselo a la variable A». Y «A» se queda cargada con 2 que es el primer dato de la sentencia DATA. En ese mismo instante, una especie de dedo pasa a señalar al dato siguiente (al 3), presto a ser adjudicado a la siguiente variable que esté detrás de otro READ. En nuestro caso, eso ocurre con la siguiente instrucción (READ B), y así sucesivamente.

Vamos a hacer algunas diabluras. Supongamos que se nos ha olvidado incluir una línea con sentencias DATA. ¿Qué sucederá tras el RUN? Pues que al intentar ejecutarse el READ, al no encontrar ningún DATA, saltará el «Out of data error», traducible libremente por «Oigame, no encuentro un dato que llevarme a la variable». Razonable gemido del programa.

Ahora vamos a incluir en la línea de DATA —dondequiera que ella esté— otro dato, v.g.:

```
400 DATA 2,3,5,7
```

Y lo demás igual. ¿Resultado? El mismo que si no hubiésemos añadido nada. «A» seguirá cargado con 2, «B» con 3 y «C» con 5. El dedo (le llamaremos «apuntador» que parece más serio) se quedará apuntando inútilmente a un siete que nadie aprovechará.

Y si dejamos como estaba al principio el DATA, con 3 datos y añadimos a la línea 20 otro READ:

```
20 READ A: READ B: READ C: READ D
```

En ese supuesto el dedo-apuntador llegará al final cuando la variable C se cargue con el último dato (el 5). Cuando, a continuación el READ D intente buscar el siguiente dato, se encontrará con que no queda ninguno, saltando el mismo error que en la primera diablura.

Alguien podrá preguntar ¿y no hay manera de «restaurar» los datos, es decir, de que el apuntador vuelva al principio? Buena pregunta. Sí, hay una sentencia para ello: es la RESTORE (Restaurar, rehacer) .

RESTORE puesta así, a secas, restaura de un modo absoluto los DATA que pueda tener el programa, o sea que el apuntador se sitúa en el *primer* dato de la *primera* línea que haya en el programa que empiece con un «DATA».

Lástima, pensará alguno. Porque lo bueno sería poder llevar el apuntador a un dato, o a una línea de datos determinada, no al comienzo absoluto. Nada más fácil. Todo está previsto. Basta con poner a la derecha del RESTORE, el número de línea donde queremos que el apuntador se coloque: RESTORE 280 o RESTORE 7000 o RESTORE 4. Así de sencillo.

Si la variable del READ es alfanumérica y el DATA que se encuentra es numérico, nada raro sucederá. El dato numérico se convierte en alfanumérico y en paz. Lo contrario en cambio da error: si el READ intenta meter en una variable numérica un dato alfanumérico, es evidente que no puede.

Los DATA alfanuméricos pueden ir entrecomillados o no, es indiferente. En lo único que puede repercutir es en la existencia o no de espacios en blanco:

```
100 DATA "FLORES",FLORES
120 READ A$:READ B$
```

Cargará en A$ exactamente lo mismo que en B$. Pero:

```
200 DATA "  FLORES  ",FLORES
210 READ A$: READ B$
```

Cargará en A$ una cadena de 10 caracteres (los dos blancos de cabeza y de cola «cuentan) y en B$ una cadena de 6 caracteres (estrictamente los de la palabra leída).

Queda algo por añadir: pueden utilizarse las comas cuando se trata de leer (READ) más de una variable:

```
300 READ A: READ B: READ X: READ Y
```

se puede poner, más brevemente:

```
300 READ A,B,C,D
```

PR-1.7: El READ-DATA, con un cierto ringorrango.

```
10 RESTORE 900
20 READ E$(1),E$(2),E$(3),E$(4)
40 E$="ESTACIONES"
50 RAYA$="----------"
70 RESTORE 880: READ A,B,C
80 DATA PEPE,4,LUIS,4,CARLOTA,7,MERCEDITAS,10
90 RESTORE 80:READ N$(1),N(1),N$(2),N(2),N$(3),N(3)
100 READ N$(4),N(4)
120 PRINT E$: PRINT RAYA$
140 PRINT E$(1):PRINT E$(2):PRINT E$(3):PRINT E$(4)
150 PRINT N$(1):PRINT N$(2)
160 PRINT N$(3):PRINT N$(4)
170 PRINT N(1):PRINT N(2):PRINT N(3):PRINT N(4)
180 PRINT A:PRINT B:PRINT C
880 DATA 1985,1515,1492
900 DATA PRIMAVERA," VERANO","  OTONO","   INVIERNO"

RUN <CR>
ESTACIONES
----------
PRIMAVERA
 VERANO
  OTONO
   INVIERNO
PEPE
LUIS
CARLOTA
MERCEDITAS
 4
 4
 7
 10
 1985
 1515
 1492
Ok
```

3.— ASIGNACION A TRAVES DE LA SENTENCIA «INPUT»

Esto son palabras mayores. Mucha atención.

La sentencia «INPUT» es una de las pocas que permiten el diálogo ordenador-operador.

En los dos modos de asignación que acabamos de ver, los valores que tomaban las variables *ya estaban*, ya están incorporados al programa. El RUN ponía en marcha la ejecución de unas instrucciones, gracias a las cuales las variables tomaban unos valores predeterminados. Cada variable «valía» lo que el *programador* había decidido que valiese. Con la sentencia INPUT, en cambio, la variable que le acompaña «valdrá» lo que el *operador* decida que valga.

Se comprende —o al menos se intuye— la riqueza que esta sentencia proporciona a los programas. Es una ventana abierta a la iniciativa de alguien distinto a quien creó el programa. Más aún, el operador puede no tener la menor idea de «cómo» es el programa por dentro y sin embargo, trabajar con él con toda soltura haciéndole recorrer caminos muy distintos a los que el programador pudiera imaginar. (Atención: hemos escrito «imaginar», no «prever»: porque el programador ha de preverlo *todo*, oiga, todo. A veces se oye decir a *operadores* quizás algo timoratos: «He debido de hacer algo mal porque me ha saltado un error». No. Si ha saltado un error, el que ha hecho algo mal es el programador, quien, insistimos, *tiene que preverlo todo,* incluso que ante el teclado se siente un chimpancé y se ponga a monotear con las teclas. Ni aún así tiene que quedar abortado un programa por haber aparecido un error. Tómese buena nota de ello.)

PR-1.8: Primeras escaramuzas con INPUT.

```
10 PRINT "Dime como te llamas:"
20 PRINT
30 INPUT N$
60 PRINT
70 PRINT "Dame un numero:"
80 INPUT N1
90 PRINT
100 PRINT "Dame otro numero:"
110 INPUT N2
120 PRINT
130 PRINT "Te voy a dar su suma, su diferencia,"
135 PRINT "su producto y su cociente:"
140 PRINT N1+N2
150 PRINT N1-N2
160 PRINT N1$N2
170 PRINT N1/N2
180 PRINT
190 PRINT "Hasta la vista,"
200 PRINT N$
```

```
RUN <CR>
Dime como te llamas:

?
```

(El cursor se quedará junto al signo de interrogación. Parece que el programa se ha parado. Claro. Se ha tropezado con el primer INPUT, y está esperando, sencillamente, a que el operador *introduzca* (escriba en el teclado) su nombre. Esta espera no tiene límite. La paciencia del ordenador es infinita. Escribimos. Y cada carácter se incorporará a la variable que acompaña al INPUT —N$ en nuestro caso— pero no *mientras* lo escribimos, sino cuando terminamos de hacerlo. Y la tecla destinada a avisar de que hemos acabado de teclear el nombre es, cómo no, CR. En el preciso momen-

to que pulsamos CR, N$ se habrá convertido en la cadena alfanumérica de que se compone nuestro nombre.)

Otra vez:

```
RUN <CR>
Dime como te llamas:

?CARLOS

Dame un numero:
?12

Dame otro numero:
?10

Te voy a dar su suma, su diferencia,
su producto y su cociente:

 22
 2
 120
 1.2

Hasta la vista,
CARLOS
Ok
```

Hacemos notar que lo que está escrito detrás de «?» en la ejecución del programa anterior, es lo que el operador ha introducido por el teclado (a instancias del programa).

Este es el tercero e importantísimo modo de asignación de valores a las variables. De él queda todavía bastante que decir, pero lo dejaremos para un poco más adelante. Y también dejaremos para después (Segunda Parte) la cuarta forma de asignación: la que tiene lugar en el manejo de Ficheros. Porque ahora nos urge hacer un inciso: es llegada la hora de dominar la *puntuación.*

PUNTUACION

Por puntuación entendemos, lo mismo que en ortografía, los cuatros signos siguientes:

Veamos la función que desempeña cada uno dentro del Basic.

El *punto (.)* ya nos es bien conocido. Es el punto decimal (equivalente a «la coma» que utilizamos en los países latinos). Ese es su principal —que no único— empleo en Basic. Cuando un número no entero es menor de la unidad, se puede expresar con o sin el «cero» de su parte entera.

Por ejemplo, si pretendemos asignar a la variable A «medio kilo», lo primero que hemos de saber es que —como variable numérica que es— «A» nada va a retener de lo del «kilo». Guardará dentro de sí estrictamente el valor numérico. Nada sabrá de la unidad de medida a la que el número se refiere.

Podremos poner A = 0.5 ó A = .5 (es lo mismo).

Pero si es el ordenador (el programa) el que responde, se ahorrará el «0» inicial.

Es decir, si escribimos:

```
PRINT 1/2
```

Saldrá: .5

No tan importante —pero necesario saberlo— es el empleo del punto como carácter integrante de un nombre de variable:

```
PUNTO.DE.PEDIDO=200
CODIGO.POSTAL$="28007"
```

Si se nos ocurriera hacer la (falsa ella) asignación:

```
120 PUNTO DE PEDIDO=200
```

saltaría un estridente «Syntax error in 120». Es natural, porque los espacios en blanco no son caraceres admisibles para denominar una variable. En casos como ése es el punto lo que mejor encaja.

Los *dos puntos (:)* también nos resultan familiares. Hacemos uso de ellos principalmente para *separar* dos instrucciones dentro de una misma línea. Cuando escribimos:

```
200 X=1: Y=2: Z=3
```

En realidad hemos puesto tres instrucciones dentro de la línea 200. Son tres asignaciones independientes. Los dos puntos actúan de barreras separadoras e impermeables.

Y como uso menos frecuente, los dos puntos se emplean también (lo veremos en su momento) para definir el disco al que nos dirigimos para llamar a un programa o un fichero. Por ejemplo, al

escribir, «B:PEDIDOS» nos estamos refiriendo al fichero de PEDIDOS que se aloja precisamente en el disco o diskette «B» (puede que haya otro con el mismo nombre en el «A»).

El *punto y coma (;)* se utiliza siempre con las sentencias PRINT e INPUT.

Como ya nos habremos dado cuenta, al escribir PRINT LOQUESEA sin más aditamento, se produce una aparición en pantalla de LOQUESEA seguida de una «caída» del cursor a la línea inferior. La línea:

```
100 PRINT A: PRINT B: PRINT C
```

produce una salida en pantalla de los valores A, B y C *cada uno en una línea.*

Si queremos que los tres salgan en la misma línea, escribiremos:

```
100 PRINT A;: PRINT B;: PRINT C
```

Y —mucho mejor— conseguiremos idéntico resultado poniendo:

```
100 PRINT A;B;C
```

El punto y coma tiene la misión, pues de *sujetar* el cursor en la línea en que éste se encuentre. Imaginémoslo desde ya, como una especie de gancho que enlaza los sucesivos «PRINT».

En múltiples programas posteriores veremos profusamente la función de nuestro punto y coma.

La *coma (,)* por último, tiene dos empleos importantes:

En pantalla: funciona de modo parecido al punto y coma sólo que produce una «tabulación» automática de lo que se trata de reproducir mediante el PRINT.

Por ejemplo:

```
100 PRINT A,B,C
```

sacará los valores de A, B y C, en la misma línea pero con una separación (tabulación) estándar entre ellos de 15, 10, 12 espacios, según el ordenador.

Fuera de esta función tabuladora, la coma puede considerarse el *separador* por excelencia. Recientemente lo hemos visto en las sentencias DATA y READ. Cada dato del DATA y cada variable del READ quedaba separado del (de la) siguiente por una coma. Irán saliendo otras distintas funciones de separador para la coma.

PR-1.9: El mismo programa PR-1.8 pero con un formato algo más estético:

```
10 PRINT "Dime como te llamas: ";
20 INPUT N$
30 PRINT
40 PRINT "Dame un numero: ";
50 INPUT N
60 PRINT: PRINT "Dame otro numero: ";
70 INPUT M
80 PRINT
90 PRINT "Esta es su suma:";N+M
100 PRINT "Su diferencia:  ";N-M
110 PRINT "Su producto:    ";N*M
120 PRINT "Su cociente:    ";N/M
130 PRINT
140 PRINT "Hasta luego, ";N$

RUN <CR>
Dime como te llamas: ?CARLOS

Dame un numero: ?36

Dame otro numero: ?9

Esta es su suma: 45
Su diferencia:   27
Su producto:     324
Su cociente:     4

Hasta luego, CARLOS
Ok
```

Ya va quedando mejor. Se echa de menos, tal vez, que el propio INPUT no disponga de la posibilidad de contener mensajes por sí mismo, sin tener que recurrir siempre a PRINTs. Pues sí señor. El INPUT dispone de la posibilidad de llevar mensajes incorporados. Muy útil. Veámos cómo.

PR-1.10: El programa anterior mejorado y cuyo RUN daría un resultado idéntico al anterior.

```
10 INPUT "Dime tu nombre: ";N$
20 PRINT
30 INPUT "Dame un numero: ";N
35 PRINT
40 INPUT "Dame otro numero: ";M
50 PRINT
60 PRINT "Esta es su suma:";N+M
70 PRINT "Su diferencia:  ";N-M
80 PRINT "Su producto:    ";N*M
85 PRINT "Su cociente:    ";N/M
90 PRINT
95 PRINT "Hasta luego, ";N$
```

Fíjese el lector que las líneas 20, 35, 50 y 90 constituidas por un simple PRINT no sirven más que para producir una línea en blanco. Pura cuestión de estética.

REM

Sentencia sencilla e inofensiva entre todas. Esas tres letras, REM, abreviatura de REMARK (= anotación), puestas al comienzo de una instrucción no producen absolutamente *ningún* efecto explícito en la ejecución del programa. ¿Entonces para qué sirve el REM, si puede saberse? Pues para escribir tras ellas todo cuanto queramos: todo cuanto nos pueda facilitar la comprensión de lo que hace el programa. Es una sentencia para el programador, no para la máquina, que la desprecia olímpicamente.

En la ejecución del programa, todo cuanto pueda haber tras un REM, es ignorado por aquél, pasando el control a la *línea* siguiente.

Resumiendo, los REM, seguidos de los correspondientes mensajes, constituyen la *documentación* del programa. ¿Y qué se puede escribir detrás de un REM? Pues mire Vd. exactamente *lo que quiera,* siempre que no se pase de la longitud de la línea. Asimismo, Vd. puede emplear todos los REMs que le parezca.

PR1.11: Utilización del REM.

```
10 REM ESTE PROGRAMA DA EL VOLUMEN DE UN CILINDRO,
20 REM PARA LO QUE LE PIDE AL OPERADOR LOS DATOS
30 REM NECESARIOS, ES DECIR, EL RADIO DE LA BASE Y
40 REM LA ALTURA DEL CILINDRO.
50 REM
60 INPUT "RADIO DE LA BASE: ",R:IF R=0 THEN END
65 REM SI TECLEAMOS 0 SE TERMINA EL PROGRAMA.
70 INPUT "ALTURA DEL CILINDRO: ",H
80 PRINT "VOLUMEN = ";3.1416*R^2*H
90 GOTO 60
100 REM AQUI SE ACABA EL LISTADO DEL PROGRAMA.
```

En este programa, sólo las líneas 60, 70 y 80 producen un efecto práctico. Por todas las demás, debido a los REM, la ejecución «resbala» sin producir efecto alguno.

En el BASIC de Microsoft se puede sustituir con idéntico resultado, la palabra REM por una comilla simple. Así pues, el programa anterior también se podría escribir:

```
10 ' ESTE PROGRAMA DA EL VOLUMEN DE UN CILINDRO,
20 ' PARA LO QUE LE PIDE AL OPERADOR LOS DATOS
30 ' NECESARIOS, ES DECIR, EL RADIO DE LA BASE Y
40 ' LA ALTURA DEL CILINDRO.
50 '
60 INPUT "RADIO DE LA BASE: ",R:IF R=0 THEN END
65 ' SI TECLEAMOS 0 SE TERMINA EL PROGRAMA.
70 INPUT "ALTURA DEL CILINDRO: ",H
80 PRINT "VOLUMEN = ";3.1416*R^2*H
90 GOTO 60
100 ' AQUI SE ACABA EL LISTADO DEL PROGRAMA.
```

CAPITULO 4.— BIFURCACIONES Y CONDICIONES

Estudiamos en este capítulo las sentencias siguientes:

- GOTO
- IF .. THEN
- ELSE
- ON
- GOSUB

Entremos con entusiasmo en el estudio de las instrucciones más importantes del Basic y de cualquier lenguaje de alto nivel. Son las sentencias que hacen que el ordenador se comporte «casi» como un ser inteligente, las que le dan capacidad de «decisión» dicho sea sin ánimo de ofender a psicólogos y moralistas.

Antes, cuando decíamos que el Basic ejecuta las instrucciones que encuentra, siguiendo un número de línea creciente, no estábamos diciendo estrictamente la verdad. Porque eso es cierto *mientras no haya una sentencia,* dentro del propio programa, *que le obligue a cambiar el rumbo.* Y ésas son las sentencias de bifurcación.

GOTO

La primera, la más elemental es la *GOTO*. En mal inglés eso significa «VE» o «VETE A» (en buen inglés sería GO TO). Eso de VETE A, ciertamente suena un poco a insulto. Pero GOTO, seguido de un número de línea produce, sencillamente, el paso a la ejecución de la línea que le acompaña. Ese «paso» o «salto» (esa bifurcación) puede ser hacia adelante o hacia atrás. Hacia un número de línea inferior o superior a aquel donde se encuentra el GOTO. A *cualquier* número de línea, con la única condición de que esa línea exista.

Entonces:

```
GOTO 1000
```

así, a palo seco, es una orden *incondicional* de que el programa siga ejecutándose a partir de la(s) instrucción(es) contenida (s) en la línea 1000.

PR-1.12: Primer programa con GOTO.

```
10 PRINT "CUANTAS VECES TENDRE QUE REPETIRTELO?"
20 GOTO 10
```

Tratemos de averiguar lo que va a suceder, cuando le demos RUN. Empieza la ejecución de la línea 10. En pantalla aparecerá el mensaje de marras. A continuación se ejecuta la instrucción de la línea 20: «GOTO 10», con lo que se vuelve a ejecutar la línea 10. Luego pasa a la 20, que le envía a la 10... ¿y así hasta cuando? Bueno, digamos que hasta siempre.

```
RUN <CR>
CUANTAS VECES TENDRE QUE REPETIRTELO?
CUANTAS VECES TENDRE QUE REPETIRTELO?
CUANTAS VECES TENDRE QUE REPETIRTELO?
CUANTAS VECES TENDRE QUE REPETIRTELO?
- - - - -
```

Hemos provocado un *bucle cerrado.* No hay posibilidad de escapar. El mensaje seguirá apareciendo hasta que se arbitre uno de los siguientes drásticos remedios:

— Aporrear el ordenador con una maza contundente.

— Llamar a la Compañía Eléctrica para que corte la luz.

— Desconectar la máquina.

— Dar CONTROL C con lo que abortaremos el programa.

(Esperamos que nuestro lector, en semejante tesitura, opte por la última de ls soluciones como menos violenta.)

PR-1.13: Otro bucle cerrado pero algo menos monótono.

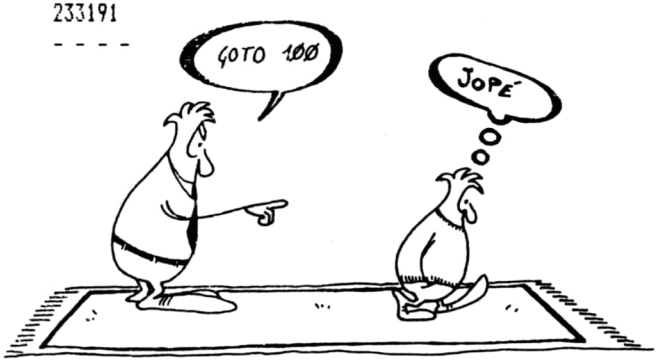
```
10 X=X+1
20 PRINT X
30 GOTO 10

RUN <CR>
 1
 2
 3
 4
- - - -
```

Recordemos que el valor de X, en el comienzo absoluto del programa, vale 0, como todas. La ejecución de la línea 10 hace que

X pase a valer 1. Llega a la 20 y aparece un «1» en la pantalla. Luego, la 30, bifurca, quieras o no, a la línea 10. Estamos como al principio, pero con la única diferencia que «X» ya no vale 0 sino 1 cuando vuelve a la línea 10. Y a la salida de la misma, X pasa a valer 2 y así, dale que te pego.

Si le damos RUn y nos vamos al cine, cuando volvamos veremos algo así en la pantalla:

```
233187
233188
233189
233190
233191
- - - -
```



Un contínuo «scrolling». También en este caso estamos metidos en un círculo completamente vicioso. Cuando nos cansemos de ver salir números correlativos, damos CONTROL C y se acabó.

PR-1.14: Variaciones sobre el mismo tema.

```
10 N=N+2
20 PRINT N;
30 GOTO 10

RUN <CR>
 2  4  6  8  10  12  14  16  18  ....
```

Aparecerán los números pares llenando pantallas y más pantallas. Obsérvese el efecto causado con la simple inclusión del punto y coma en la línea 20.

Si sustituimos el punto y coma por la coma, el resultado es parecido: los números salen en varias columnas ordenadas gracias al efecto producido por la coma.

PR-1.15: Programa cuya ejecución no sirve absolutamente para nada pero que es, en cambio, inútilmente complicado.

```
10 CUENTA=CUENTA+7
20 PEPE=PEPE↑2+1
30 GOTO 100
40 PRINT PEPE,
50 B=B-1
60 PRINT B^2,
70 GOTO 10
100 CUENTA=CUENTA-3
110 PRINT CUENTA,
120 GOTO 40

RUN <CR>
 4          1          1          8          3
 4          12         7          9          16
 15         16         20         31         25
 24         63         36         28         127
 - - - - -
```

Este sería un bucle también cerrado aunque de múltiples lazos.

(Pronto daría, si antes no lo hemos abortado, «Overflow in 20»).

Se masca la necesidad de inventar algo que nos saque de estos círculos viciosos sin tener que recurrir a la Cía. Eléctrica.

IF

Ya está inventado. Es la fabulosa sentencia *IF*, la más importante no sólo del Basic sino de cualquier lenguaje de cualquier nivel (aunque en los de bajo nivel no aparezca exactamente como «IF»).

Podemos afirmar sin exageración, que sin el «IF» no existiría la informática.

Es la sentencia que genera toda la riqueza que un buen programa puede llegar a alcanzar. Es el «SI» condicional de nuestro catellano. El «si ocurre esto, entonces hago aquello» de nuestro libre albedrío. Es el «IF» de Rudyard Kipling. Gracias al IF las criaturas se acercan peligrosamente a sus creadores.

Vamos con él.

Su forma más sencilla es:

```
IF (Expresión lógica cualquiera) THEN Instrucción(es)
```

Englobamos en el término «expresión lógica» todas las enumeradas en el capítulo anterior (tanto las estrictamente lógicas como las relacionales).

Empecemos por lo más facilón.

PR-1.16: Programa que saque en pantalla los 18 primeros números naturales.

Primero dos procedimentos a cuál más ingenioso (para tal viaje no necesitábamos alforjas):

```
10 PRINT 1
20 PRINT 2
30 PRINT 3
- - -
180 PRINT 18
```

```
10 PRINT "1"
20 PRINT "2"
30 PRINT "3"
- - -
180 PRINT "18"
```

Sin comentarios.

3.er procedimiento:

```
10 A=1
20 PRINT A
30 A=A+1
40 IF A>18 THEN GOTO 100
50 GOTO 20
100 END
```

Esto es otra cosa. Hemos aprendido de paso una sentencia tan obvia que se explica por sí sola. Es la que aparece en la línea 100. Hasta ahora los programas terminaban por consunción: porque llegaban al final y allí concluían al no encontrar más instrucciones. En realidad, en esos casos el Basic ejecuta un verdadero END que le permite al operador-programador recuperar el modo control.

Veamos cómo funciona el programa. Hasta la 30 todo va bien. Al llegar a la 40, la variable A vale 2. Como A (o sea, 2) *no es* mayor que 18, el programa sigue en la *línea siguiente,* la 50, que bifurca incondiconalmente a la 20, etc.

En el bucle que hay entre la 20 y la 50 el programa se queda dando vueltas hasta que A alcanza el valor 19. Esa es la primera vez que se cumple la condición del IF. Porque 19 es el primer valor que recibe A (línea 30) que resulta ser mayor de 18. Por consiguiente, al *cumplirse* la condición del IF, pasa a ejecutarse lo que hay a continuación del THEN, es decir, el fatídico «GOTO 100» donde el programa llega a su muerte natural.

La traducción al castellano de la línea 40, que es la madre del cordero, sería:

«Si el valor de A es mayor de 18, entonces vete a la línea 100»

Variante al 3.er procedimento:

```
10 A=1
20 PRINT A
30 A=A+1
40 IF A>18 THEN END
50 GOTO 20
```

La línea 40 se traduciría ahora:

«Si el valor de A es mayor de 18, entonces *termina*».

con lo cual produce el mismísimo resultado que antes.

4.º Procedimiento: (vayamos acostumbrándonos a las líneas multi-instrucción)

```
10 A=A+1:PRINT A:IF A=18 THEN END
20 GOTO 10
```

Recomendamos al lector que intente sacar un par de variantes por su cuenta: es saludable. Por ejemplo cambiando la situación del IF.

PR-1.17: En lugar de loa 18 primeros números, queremos que aparezcan los «N» primeros números, siendo N el número que le venga en gana al operador.

```
10 INPUT "Hasta que numero quieres que saque?: ";LIMITE
20 N=N+1
30 IF N>LIMITE THEN END
40 PRINT N:GOTO 20
```

PR-1.18: Igual que antes, pero con la oportunidad de pedirle nuevas ristras de números (esta vez, además, uno al lado del otro).

```
10 PRINT
20 INPUT "Hasta que numero quieres que saque?: ";LIMITE
30 N=0
40 N=N+1
50 IF N>LIMITE THEN 10
60 PRINT N;:GOTO 40
```

Tres comentarios:

— La línea 15 es necesaria. Es lo que se entiende como «inicialización de variable». Si no la pusiéramos, el valor de N se acumularía de una a otra vez, impidiendo conseguir las listas deseadas. Insistiremos más tarde en el tema de las inicializaciones.

— Con toda intención hemos puesto al final de la 40 «... THEN 10» en lugar de «THEN ... GOTO 10». Es legítimo. Cuando el cumplimiento de un IF lleva a una bifurcación caben tres formas de expresarlo:

```
IF N>LIMITE THEN GOTO 10
IF N>LIMITE THEN 10
IF N>LIMITE GOTO 10
```

Que no sea por falta de facilidades.

— Nos hemos vuelto a meter con este programa en un callejón sin salida. No podemos terminarlo sin recurrir a la violencia por inofensiva que ésta sea. Mal hecho. Siempre hay que dejar una salida, un modo de escapar.

PR-1.19: Salida de números correlativos, a voluntad del operador, con escape del mismo.

```
10 PRINT
20 INPUT "Hasta que numero quieres que saque? (0=FIN): ";LIMITE
30 IF LIMITE=0 THEN END
40 N=0
50 N=N+1
60 IF N>LIMITE THEN 10
70 PRINT N;:GOTO 50
```

PR-1.20: Programa completo de salida de números por pantalla, dejando a merced del operador tanto los intervalos como el sentido (creciente o decreciente):

```
10 INPUT "Desde que numero quieres? (0=Fin): ";INF
15 IF INF=0 THEN END
20 INPUT "Hasta que numero?";SUP
30 IF SUP<INF THEN PRINT "NO DIGAS TONTERIAS":GOTO 10
40 INPUT "Al derecho (D) o al reves (R)?: ";X$
50 IF X$<>"D" AND X$<>"R" THEN PRINT "PULSA D o R":GOTO 40
60 IF X$="R" THEN 200
70 N=INF
80 PRINT N:N=N+1:IF N>SUP THEN 10
90 GOTO 80
200 N=SUP
210 PRINT N:N=N-1:IF N<INF THEN 10
220 GOTO 210
```

Comentarios:

En la línea 30 protegemos la superioridad del límite «mayor» frente al «menor».

En la línea 50 establecemos dos condiciones de comparación ligadas por una condición lógica. Significa: «si el contenido de X$ es distinto de «D» *y* distinto también de «R», entonces que aparezca el mensaje etc.».

El seguimiento del resto del programa lo dejamos en las buenas manos del lector.

Las posibilidades del IF son inconmensurables. Veamos unas cuantas.

```
-IF A=B THEN Z=X+Y:L=2:READ M$:GOTO 2290

-IF CONTADOR=0 THEN CONTADOR =1

-IF DIA<1 OR DIA>31 OR MES<1 OR MES>12 THEN PRINT "HORRIBLE
 FECHA, DIGA":GOTO 1200

-IF DIA>29 AND MES=2 OR DIA>30 AND MES=4 THEN 440
```

Anotamos:

Si la condición (o condiciones) del IF se cumplen, se ponen en marcha todas las instrucciones que le siguen dentro de la línea. En el primer ejemplo, «si A = B entonces (todo lo que sigue)». y si A no fuese igual a B se saltarían todas las instrucciones que hay en esa línea, detrás del IF.

El tercer ejemplo se traduciría por: «si DIA es menor de 1 *o* DIA es mayor de 31 *o* MES es menor de 1 *o* etc.»

El número de condiciones que pueden incluirse con un IF sólo queda limitado por los 254 caracteres que una línea de instrucciones admite.

En los casos de «multicondición» (como son el tercero y el cuarto), no vale omitir nada, cada condición ha de escribirse completa. Por ejemplo, en el caso 3, no valdría poner:

```
IF DIA<1 OR >31 ....
```

Es preciso repetir la variable DIA.

El AND se ejecuta antes del OR. Cuando se incluyen varios OR y AND mezclados en una multicondición, hay que tener esto presente para, en caso necesario, emplear paréntesis si fuera preciso, exactamente igual que si se tratara de una expresión algebraica en la que el AND fuese «*» (producto) y OR fuese «+» (suma). De hecho, en las últimas versiones del Basic de Microsoft, se puede utilizar en las expresiones lógicas, el «+» en lugar del OR y el «*» en sustitución del AND.

Así, en el ejemplo 4, y aunque no es necesario poner paréntesis, las cosas funcionan «como si» la instrucción se hubiese escrito así:

```
-IF (DIA>29 AND MES=2) OR (DIA>30 AND MES=4) THEN 440
```

Que también podemos poner (separando con espacios inofensivos para que se vea mejor):

```
-IF (DIA>29) * (MES=2) + (DIA>30) * (MES=4) THEN 440
```

ELSE

Para terminar con el poderoso IF, estudiemos la sentencia ELSE, muy útil ella, y que acompaña con frecuencia a las instrucciones condicionales.

Dijimos antes que cuando aparece una sentencia con «IF», el *no* cumplimiento de la(s) condición(es) produce el salto a la siguiente LINEA. Bien, esto es completamente cierto, excepto si hay, dentro de la línea del IF un ELSE que haga cambiar la lógica de las cosas.

ELSE significa aquí: «EN OTRO CASO» o «SI NO SE CUMPLE LA CONDICION».

Ejemplo:

```
100  IF  DIA$="MARTES" AND DIA=13 THEN PRINT  "DIA  NEFASTO"
         ELSE PRINT "DIA FASTO"
110  GOTO 70
```

Una de dos, o es martes y 13 o no lo es. Si lo es, aparecerá el mensaje «DIA NEFASTO» y no hará ningún caso a lo que el ELSE pueda llevar detrás. Y si no lo es, se saltará lo que acompaña inmediatamente al IF y pasará a ejectuarse lo que haya tras el ELSE («DIA FASTO»). En ambas alternativas, el programa continuará en la línea 110 (es decir, el control del programa pasará a la línea 70).

Esto, que no parece tener mucho misterio, puede complicarse sobremanera en la práctica:

```
300  IF A=1 THEN 185 ELSE IF A<0 THEN B=8:C=4:GOTO 1100
     ELSE 1200
```

Con cuentagotas: si A es igual a 1, no hay duda de que el programa bifurcará a la línea 185.

Si no se cumple esa condición entra en funcionamiento lo que hay detrás del primer ELSE que a su vez (¡horror!) es otra condición. No hay que alarmarse. Ahí leemos que si A es negativo entonces después de asignar 8 a B y 4 a C, el programa salte a la línea 1100. Todo claro.

Pero ahora viene el lío. Otro ELSE. Que no cunda el pánico. ¿A quién corresponde? ¿Al no cumplimiento del primero o del segundo IF? Pues naturalmente que al segundo, porque el primero ya tenía su ELSE.

Si el programa entrara 4 veces en la 300, cada vez con uno de estos valores para la variable A:

```
-12, 0, 1, 3
```

el programa bifurcará respectivamente a las líneas 1100, 1200, 185, 1200. Sígase el proceso lógico, por favor.

Vamos ahora con el caso más enrevesado. Nos encontramos con esta instrucción:

```
320 IF A>0 THEN IF A=1 THEN IF B=0 THEN 200 ELSE 220 ELSE
    280 ELSE 400
```

Traduzcámosla al castellano (ánimo): «Si A es un número positivo, en ese caso, si ese número es el 1 entonces, si B = 0 el programa se bifurca a la línea 200; si no es cero B (pero A = 1) entonces se bifurca a la 220; y si A no era igual a 1 en ese caso salta a la 280; y por último, si ni siquiera A es positiva, se va el programa a la línea 400». Clarísimo. Vaya cruz. Obsérvese cómo la correspondencia entre los IFs y sus ELSEs tiene lugar a modo de nido (el primero con el último, el segundo con el penúltimo, etc.)

(Ocasión de oro para traer a colación un retazo de la joven sabiduría informática.)

Los americanos, pragmáticos y sensuales ellos, recomiendan siempre al programador el «KISS», iniciales de «Keep It Simple, Stupid» que traducimos libremente por «No te metas en líos, no seas bobo y haz sencillos los programas». Nunca olvidemos aquello de que «el mejor programa es el que mejor funciona». Nadie le va a subir a Vd. el sueldo porque su programa tenga un precioso aspecto de gigantesco comecocos. Siempre hay un modo de programar más claro sin pérdida apreciable de tiempo y espacio.

La línea de instrucciones:

```
600 IF A=1 THEN 700 ELSE IF A=2 THEN 800 ELSE IF A=3 THEN
    900 ELSE IF A=4 THEN 1000 ELSE IF A=5 THEN 1100
```

expresa ya no una «bi» sino una «multi»-bifurcación en la que cualquier valor de la variable A distinto de los que en las condicio-

nes se expresan haría seguir el programa en la línea siguiente a la 600 y para valores de A iguales a 1, 2, 3, 4 ó 5 el control pasaría a las líneas 700, 800, 900, 1000 ó 1100 respectivamente.

ON

Existe una sentencia *(ON)* que hace esto mismo sin tanta pandereta, con un formato más limpio:

```
600 ON A GOTO 700,800,900,1000,1100
```

Significa: «Si A = 1 vete a la 700, si A = 2 vete a la 800, etc.» Naturalmente, si A no toma ninguno de esos valores, el programa seguirá en la línea siguiente a la 600.

Se trata, por consiguiente de un «IF múltiple», con la ventaja adicional, de que el intérprete redondea automáticametne los valores que pueda tomar A comprendidos entre 1 y 5 en el ejemplo o entre 1 y el *número de salidas* que figuren en la instrucción.

PR-1.21: Ejemplo —tan insulso como claro— de uso del ON.

```
10 INPUT K:IF K=0 THEN END
20 ON K GOTO 40,50,60,70,80
30 GOTO 10
40 PRINT "UNO":GOTO 10
50 PRINT "DOS":GOTO 10
60 PRINT "TRES":GOTO 10
70 PRINT "CUATRO":GOTO 10
80 PRINT "CINCO":GOTO 10
```

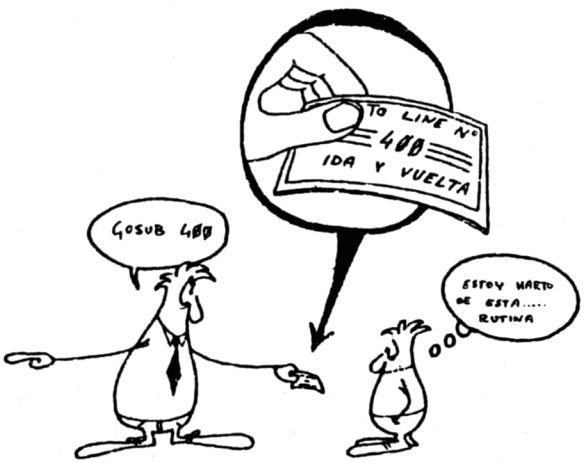
GOSUB

Y dejamos para el final de este capítulo la última aunque no por ello menos importante sentencia: la potente, la hiperutilizada, la nunca bastante bien ponderada *GOSUB.* (En un buen programa

profesional, es seguramente, la sentencia que más veces aparece).

El «GOTO (Línea)» produce una bifurcación al número de línea que acompaña al GOTO. Sólo otro GOTO amigo hará regresar al hijo pródigo (o la propia secuencia del programa). El GOTO es un viaje con billete de ida.

El *GOSUB*, en cambio, es un *vete pero vuelve*. Es un viaje con billete de ida y vuelta. La instrucción «GOSUB 80» viene a significar «Vete a la línea 80 y ejecuta el "sub" programa que allí encuentres, y cuando hayas terminado, vuelve, por Dios».



Surgen preguntas que vamos contestando en plan catecismo:

Pregunta: Eso de Subprograma, ¿qué es? ¿Algo misterioso?

Respuesta: nada de eso. Es, sencillamente, *una parte* del programa en Basic que estamos ejecutando (o que estamos confeccionando). También se le llama «rutina» o «subrutina».

RETURN

Pregunta: Dice Vd. «cuando hayas terminado, vuelve». ¿Y cómo sabe el subprograma que ha terminado? ¿Con un END tal vez?

Respuesta: No, no. Un END terminaría *todo* el programa. La palabra mágica que obliga a regresar al punto de partida es precisamente «vuelve», «retorna», pero en inglés. O sea, *RETURN.* RETURN es, como el END, una sentencia absolutamente solitaria, es decir, que no va acompañada de argumento ni expresión ninguna. Es así, a secas RETURN.

Pregunta: ¿Y adónde «retorna» exactamente?

Respuesta: a la *instrucción siguiente* al GOSUB que produjo la bifurcación. Atención, no a la «línea siguiente»; repetimos, a la instrucción siguiente (que puede estar en la línea siguiente o en la misma línea).

```
560 GOSUB 1000
570 BLA, BLA
580 BLA, BLA
- - -                          1000 BLA, BLA
700 BLA,BLA                    1010 BLA, BLA
710 GOSUB 1000                 1020 BLA, BLA
720 BLA, BLA                   1030 RETURN
730 BLA, BLA
- - -
```

No será raro encontrar en un programa esta línea que ahora puede parecer terrible:

```
440 GOSUB 80:GOSUB 110:GOSUB 42:IF U=0 THEN GOSUB 100
```

Cuando el programa llega a la línea 440, se produce una bifurcación a la 80 hasta que encuentra un RETURN, con lo que regresa a la 440, donde a su vez se encuentra con otro GOSUB que le envía a la 110, etc.

```
                               42 BLA, BLA
440 GOSUB 80:                  50 BLA, BLA
    GOSUB 110:                 70 RETURN
    GOSUB 42:                  .....
    IF U=0 THEN GOSUB 100      80 BLA, BLA
                               85 RETURN
                               .....
                               100 BLA, BLA
                               110 BLA, BLA
                               .....
450 BLA, BLA                   140 RETURN
```

Y aún puede complicarse la cosa. Porque de un subprograma/rutina puede salir otro subprograma/rutina:

```
800 BLA, BLA
810 GOSUB 5000
820 BLA, BLA
- - -
```

```
5000 BLA, BLA
5010 BLA, BLA
5020 GOSUB 6000
5030 BLA, BLA
5040 BLA, BLA
5050 RETURN
- - -
```

```
6000 BLA, BLA
6010 BLA, BLA
6020 RETURN
- - -
```

Esto es lo que se llama una «rutina anidada». La lógica no puede ser más estricta. Las líneas 800s hacen de programa principal respecto a las 5000s. Y éstas a su vez hacen de programa principal respecto a la rutina que reside en las 6000s.

Este anidamiento o subordinación encadenada puede tener tantos escalones como nos sean necesarios. O sea, que de la rutina 6000 podría salir otra a la 7000, y así sucesivamente.

Pregunta obvia: ¿Y qué necesidad hay de complicarse la vida, de inventar nada nuevo, si el GOTO, con ponerle otro GOTO de vuelta ya hace lo mismo que el GOSUB?

Respuesta: Sí señor, muy cierto. Pero hemos tocado el mondongo del asunto. Porque la verdadera utilidad del GOSUB todavía no se ha mencionado.

Sucede en programación, a cualquier nivel, lo que sucede en la vida misma. Que hay muchos trabajos, tareas, quehaceres, que se repiten constantemente. Desde el acto de levantarnos de la cama por la mañana hasta el de respirar o estornudar, realizamos muchas, muchísimas «rutinas». Sin ellas, el esfuerzo de vivir sería incesante y agotador. Todo cuanto vivimos o experimentamos en cuanto se repite dos, tres veces, nuestro «sistema operativo» nos lo convierte en rutina. Sencillamente, para no tener que hacer trabajar a la imaginación más de lo debido. Para no tener que inventar lo que ya está inventado.

Bueno, pues algo muy parecido ocurre cuando se está programando. Que en cuanto un grupo de instrucciones se repite es señal de que esas instrucciones deben constituir una rutina, un subprograma al que se pueda acceder desde distintas zonas del programa «principal». Y entonces ya está contestada la pregunta. El GOTO de ida y vuelta ya no funcionaría porque si bien la «ida» está claramente determinada, el GOTO de «vuelta», ¿a dónde lo llevamos si puede haber, por ejemplo, 4 procedencias distintas? El RETURN, en definitva es un «VETE allí de donde hayas venido, sea de donde fuere», que es precisamente lo que queremos.

Nada como los ejemplos para que todo quede claro:

PR-1.22: Primera escaramuza con GOSUB:

```
10 INPUT "Escribe un numero menor de 100: ",N
20 IF N>=100 THEN GOSUB 200:GOTO 10
30 INPUT "Y ahora uno que sea igual o menor de 10: ",M
40 IF M>10 THEN GOSUB 200:GOTO 30
50 INPUT "Escribe el producto de ambos numeros: ",P
60 IF P<>M*N THEN GOSUB 200:GOTO 50
70 PRINT "Bravo.":END
200 PRINT "MAL. REPITE LA ENTRADA."
210 RETURN

RUN <CR>
Escribe un número menor de 100: 112
MAL. REPITE LA ENTRADA.
Escribe un número menor de 100: 44
Y ahora otro que sea igual o menor de 10: 8
Escribe el producto de ambos números: 372
MAL. REPITE LA ENTRADA.
Escribe el producto de ambos números: 352
Bravo.
Ok
```

PR-1.23: Programa con rutina de validación de fechas, teniendo en cuenta incluso los años bisiestos.

```
10 GOTO 1000
40 MAL=0:DI=VAL(LEFT$(F$,2))
50 ME=VAL(MID$(F$,3,2))
60 AN=VAL(RIGHT$(F$,2))
70 IF DI<1 OR DI>31 OR ME<1 OR ME>12 THEN 200
80 IF ME=2 AND DI>29 THEN 200
90 IF (ME=4 OR ME=6 OR ME=9 OR ME=11) AND DI>30 THEN 200
100 IF ME<>2 THEN RETURN
110 IF AN/4=INT(AN/4) THEN RETURN
120 IF DI<29 THEN RETURN
200 MAL=1:RETURN
1000 INPUT "Fecha de tu Nacimiento (DDMMAA): ",F$
1010 GOSUB 40: IF MAL=1 THEN 1000
1020 DIO=DI:MEO=ME:ANO=AN
1030 INPUT "Fecha de Hoy: ",F$
1040 GOSUB 40:IF MAL=1 THEN 1030
1050 IF AN<ANO THEN PRINT CHR$(7):GOTO 1000
1060 PRINT "TIENES UNOS ";AN-ANO;" ABRILES."
```

Obsérvese que la rutina de control de fechas (de la línea 40 a la 200), tiene *varios* RETURNs. Es totalmente normal que una rutina pueda tener varias formas de retornar. Recomendamos al lector que profundice en este programa que tiene bastante miga.

Para terminar (en esta primera fase de aprendizaje) con nuestro buen GOSUB, digamos algo acerca de *dónde* deben situarse las rutinas. Y cuando escribimos «deben» queremos decirlo en términos de recomendación, de pura comodidad y claridad, puesto que no existe restricción alguna respecto de la ubicación de un subprograma.

En un programa corto, da lo mismo. Todo está a la vista y es fácil repasar la secuencia de las instrucciones.

En un programa largo distinguimos dos tipos de rutinas: las generales (a las que recurre *todo* el programa o incluso varios programas independientes de una misma aplicación) y las específicas (las que utiliza una *porción* determinada del programa).

Las primeras, las generales, es recomendable que se las sitúe *al principio* del gran programa. Por una cuestión de nemotecnia y rapidez de programación. Es más fácil recordar «GOSUB 20» que «GOSUB 32440» cuando la rutina se va emplear con frecuencia. Además, un «20» se escribe más deprisa que un «32440».

Las específicas, las que sólo van a utilizarse desde una parte del programa (por ejemplo los que únicamente afectan a la porción «Entrada de Pedidos»), es preferible situarlas *cerca* de ese fragmento de programa.

La primera recomendación produciría una estructura de este tipo:

```
10 DIM A(25), X$(80)
15 GOTO 1000
20 'RUTINA GENERAL 1
30 ...
40 ...
50 RETURN
55 '
60 'RUTINA GENERAL 2
70 ...
80 RETURN
- - -
1000 'EMPIEZA EL PROGRAMA PROPIAMENTE DICHO.
```

```
1010 ...
- - -
1900 'RUTINA PARTICULAR 1
1910 ...
1950 RETURN
1960 'RUTINA PARTICULAR 2
1965 ...
1990 RETURN
- - -
```

(Es corriente leer en manuales de BASIC una recomendación opuesta a la que acabamos de hacer, es decir, «procure Vd. situar las rutinas en las líneas altas del programa, a partir de la 30000 por ejemplo». Lamentamos no estar de acuerdo. No encontramos razón seria que justifique tal consejo, como no sea el ahorro del humilde GOTO 1000 —por ejemplo— inicial.)

CAPITULO 5.— FUNCIONES

MANEJO DE CONSTANTES Y VARIABLES ALFANUMERICAS

LEFT$ RIGHT$ MID$

La «cadena» es una simple sucesión de caracteres, enlazados, yuxtapuestos. Uno a continuación del otro. Todos valen. Los que se ven y los que no. Desde el código ASCII 0 hasta el 255. En total, pues, 256 caracteres diferentes.

Las funciones que necesitamos para empezar a dominar estas cadenas son las que aparecen en cabecera: LEFT$, RIGHT$, y MID$.

Sabiendo que Left = Izquierda, Right = Derecha y Mid = Medio ya nos imaginamos por dónde anda la cosa.

Veamos el empleo de estas funciones alfanuméricas siguiendo un ejemplo:

Hemos hecho una asignación para A$ así;

```
A$="NO ME DEIS CONSEJOS, QUE SE EQUIVOCARME SOLO"
```

Queremos que B$ sea igual a «NO ME DEIS CONSEJOS» mediante el empleo de A$. Aquí está:

```
B$=LEFT$(A$,19)
```

Léase: «B$ es igual a 19 caracteres por la izquierda de la cadena que contenga A$».

Y el contenido de LEFT$(A$,2) será «NO», («dos caracteres por la izquierda»).

Y el de LEFT$(A$,4) será «NO M».

Etc.

Por la derecha, con el RIGHT$, la cosa es muy parecida:

```
R$=RIGHT$(A$,7):PRINT R$
```

nos dará, directamente, «ME SOLO». Ojo, nos lo dará sin comillas. Estas son necesarias para las asignaciones; pero *lo que* contienen las variables es *sólo* los caracteres que haya entre ellas.

La asignación se traduciría ahora por «R$ es igual a 7 caracteres por *la derecha* de la cadena que contenga A$».

¿Y que contendría LEFT$(A$,J)? Y RIGHT$(A$,K)? Pues naturalmente, los J primeros caracteres de A$ en el primer caso y los K últimos en el segundo, que podrán variar con arreglo a los valores que puedan tomar J y K.

Si J por ejemplo, vale 5, el contenido de LEFT$(A$,J) sería «NO ME» y si valiese 0, LEFT$(A$,J) constaría de los «O» primeros caracteres, es decir, *nada.* Conviene saber que esta nada es una nada angustiosamente sartriana, absoluta. Porque hay como distintas categorías de «nadedad». Lo veremos. Esta es la nada total. Se expresa por ''''. Entre las comillas no hay nada. Ni siquiera un espacio. Si escribimos:

```
N$=""
```

estamos dejando a N$ sin nada. A '''' se le suele denominar «cadena nula».

Todas las variables alfanuméricas, con el RUN de un programa son («contienen») cadenas nulas al comenzar la ejecución del mismo. (Recordemos que las numéricas valían «0» al iniciarse la ejecución de un programa).

Volvamos a nuestro A$ = «NO ME DEJEIS etc.». ¿Tendría sentido pedir el contenido de RIGHT$(LEFT$(A$,10),4). Sí que lo tendría. Traducido sería: «4 últimos caracteres de la cadena formada por los 12 primeros caracteres». O sea, exactamente daría «DEIS».

Ejercítese el paciente lector en hallar:

```
1.- LEFT$(RIGHT$(A$,20),10)

2.- RIGHT$(LEFT$(RIGHT$(A$,7),7),7)

3.- LEFT$(A$,200)
```

(Decíamos antes que LEFT$(A$,0) da la cadena nula, lo cual no engendra error. Algo parecido ocurre en el tercer caso. Tanto la función LEFT$ como RIGHT$ toman los caracteres *que pueden,* sin frustraciones, si no consiguen más que parte de lo exigido.)

Vamos con la última función anunciada: *MID$.*

Seguimos trabajando sobre la misma cadena:

```
A$="NO ME DEIS CONSEJOS QUE SE EQUIVOCARME SOLO"
```

Si queremos meter en otra variable alfanumérica, B$, la palabra «QUE», «sacándola» de A$, podemos hacerlo —muy laboriosamente— utilizando LEFT$ y RIGHT$:

```
B$=LEFT$(RIGHT$(A$,23),3)      O tambien:
B$=RIGHT$(LEFT$(A$,23),3)
```

Pero resulta más cómodo poner:

```
B$=MID$(A$,21,3)
```

Que expresaríamos así: «B$ es igual a las 3 primeras letras que, a partir de la *21* tiene la cadena A$». Se entiende «a partir de la 21 inclusive».

De las tres funciones, MID$ es la que más se utiliza, con mucho. Sustituye además, casi por completo a las otras dos.

Con nuestro A$ de marras, MID$(A$,3,2) «valdría» «ME» y MID$(A$,1,19) sería igual a «NO ME DEIS CONSEJOS». Esta última cadena también es LEFT$(A$,19) como fácilmente se comprende. De una forma general podemos decir que, las expresiones:

```
LEFT$(A$,N)        y        MID$(A$,1,N)
```

son idénticas para cualquier valor de N no negativo.

Con intención, no hemos escrito «cualquier valor de N entero, positivo y menor o igual al número de caracteres que tenga la cadena», porque no tiene por qué ser así, aunque lo parezca:

La expresión MID$(A$,13.7,2.8) no da error. El Basic, por su cuenta, se encarga de truncar los números fraccionarios tomando de ellos la parte entera. Ese horror de expresión sería por tanto equivalente a MID$(A$,13,2). El intérprete Basic remedia a veces nuestras tonterías.

Como se ve, en las funciones LEFT$ y RIGHT$, necesitamos, además de la cadena madre (A$ en nuestro caso), un sólo parámetro (la longitud = el número de caracteres que tomamos de aquélla). En cambio con MID$ —lo acabamos de ver—, empleamos *dos parámetros* para definirla (el número de orden del carácter a partir del cual empezamos a contar y la longitud de lo que tomamos).

Falta saber si MID$(A$,20), así, sin el segundo parámetro, significa algo. En efecto, sí, significa simplemente que nos quedamos

con lo que haya en la cadena A$, a partir del carácter número 20 *hasta el final.* Así pues, la expresión anterior será igual, en nuestro caso a « QUE SE EQUIVOCARME SOLO» (la posición número 20 es un espacio).

LEN(A$)

Estamos utilizando la palabra *longitud* de una cadena, o de una «subcadena» (fragmento, trozo de cadena) y parece que se echa de menos la existencia de una función que defina esa longitud. Muy justo. Existe y se llama *LEN(A$)*. LEN es abreviatura de LENGTH, longitud en inglés.

Entonces, LEN(A$) será igual al número de caracteres de la repetida y presuntuosa frasecita, o sea, 41. A lo que va entre paréntesis (que, por fuerza tiene que ser una cadena alfanumérica) se le da el hombre genérico, utilizable en muchos tipos de funciones, de *argumento.*

Hacemos hincapié en que, aunque su argumento es alfanumérico, la función es *numérica,* da un número. Podemos hacer una asignación tal como:

```
L=LEN(A$)  o  L1=LEN(MID$(A$,13))
```

PR-1.24: Juegos de palabras.

```
10 INPUT "Dame tu nombre (o FIN): ",N$
20 IF N$="FIN" THEN END
30 A=1
40 PRINT MID$(N$,A)
50 A=A+1:IF A>LEN(N$) THEN 10 ELSE 40

RUN <CR>
Dame tu nombre: JOSE ANTONIO
JOSE ANTONIO
OSE ANTONIO
SE ANTONIO
E ANTONIO
 ANTONIO
ANTONIO
NTONIO
TONIO
ONIO
NIO
```

```
IO
0
Dame tu nombre: ROSA
ROSA
OSA
SA
A
Dame tu nombre: FIN
Ok
```

PR-1.25: Más juegos de palabras.

```
10 INPUT "ESCRIBE LO QUE QUIERAS (F TERMINA): ",Q$
20 N=1:IF Q$="F" THEN END
30 INPUT "DE DERECHAS O DE IZQUIERDAS (D/I) ",I$
40 IF I$="I" THEN 50 ELSE IF I$="D" THEN 60 ELSE 30
50 PRINT LEFT$(Q$,N):GOTO 70
60 PRINT RIGHT$(Q$,N)
70 N=N+1:IF N>LEN(Q$) THEN 10
80 IF I$="D" THEN 60
90 GOTO 50

RUN <CR>
ESCRIBE LO QUE QUIERAS (F termina): ROSAURA DE MI VIDA
DE DERECHAS O DE IZQUIERDAS (I/D): K
DE DERECHAS O DE IZQUIERDAS (I/D): D
A
DA
IDA
VIDA
 VIDA
I VIDA
MI VIDA
 MI VIDA
E MI VIDA
DE MI VIDA
 DE MI VIDA
A DE MI VIDA
RA DE MI VIDA
URA DE MI VIDA
AURA DE MI VIDA
SAURA DE MI VIDA
```

```
OSAURA DE MI VIDA
ROSAURA DE MI VIDA
ESCRIBE LO QUE QUIERAS (F termina): RAMONCIO
DE DERECHAS O DE IZQUIERDAS (I/D): I
R
RA
RAM
RAMO
RAMON
RAMONC
RAMONCI
RAMONCIO
ESCRIBE LO QUE QUIERAS (F termina): F
Ok
```

PR-1.26: Este programa nos da el número de veces que una letra cualquiera figura en una frase cualquiera.

```
10 PRINT
15 INPUT "ESCRIBE UNA FRASE (F=Fin): ",FRASE$
20 IF FRASE$="F" THEN END
30 NUMERO=0
40 INPUT "CARACTER A CONTABILIZAR (@=Fin): ",L$
50 IF L$="@" THEN 10
60 POSICION=1
70 IF MID$(FRASE$,POSICION,1)=L$ THEN NUMERO=NUMERO+1
80 POSICION=POSICION+1
90 IF POSICION<=LEN(FRASE$) THEN 70
100 PRINT  "APARECE";NUMERO;"VECES LA LETRA ";L$
120 GOTO 30

RUN <CR>
ESCRIBE UNA FRASE (F=Fin): VOLVERAN LAS OSCURAS GOLONDRINAS
CARACTER A CONTABILIZAR (@=Fin): O
APARECE 4 VECES LA LETRA O
CARACTER A CONTABILIZAR (@=Fin): A
APARECE 4 VECES LA LETRA A
CARACTER A CONTABILIZAR (@=Fin): @
ESCRIBE UNA FRASE (F=Fin): DE TU BALCON SUS NIDOS A COLGAR
CARACTER A CONTABILIZAR (@=Fin): M
APARECE 0 VECES LA LETRA M
CARACTER A CONTABILIZAR (@=Fin): @
ESCRIBE UNA FRASE (F=Fin): F
Ok
```

OPERACIONES CON CADENAS

El único operador «aritmético» que admiten dos cadenas es el +. Hemos entrecomillado lo de aritmético porque, por supuesto, no se trata de una suma en tal sentido. Las cadenas se «suman» de la única manera que pueden hacerlo: *adosándose* la una a la otra.

Miniprograma ejemplo:

```
10 A$="MARILYN":B$="MONROE"
20 PRINT A$:PRINT B$:PRINT A$+B$:PRINT B$+A$

RUN <CR>
MARILYN
MONROE
MARILYNMONROE
MONROEMARILYN
Ok
```

PR-1.27: Programa muy útil para cuando Vd. sea un adulto en BASIC. Se trata de cargar por teclado en NOMB$ un nombre. Y queremos garantizar que NOMB$ tenga exactamente 20 caracteres, recortando a esa cifra los que tengan de más y completando a 20 con espacios a la derecha los que tengan menos.

```
10 INPUT "NOMBRE (F=FIN): ";NOMB$
20 IF NOMB$="F" THEN END
40 NOMB$=NOMB$+"                    ":REM 20 espacios.
50 NOMB$=LEFT$(NOMB$,20)
60 PRINT NOMB$,LEN(NOMB$):GOTO 10

RUN <CR>
NOMBRE (F=FIN): PEPE PEREZ
PEPE PEREZ                  20
NOMBRE (F=FIN): REGULCIPERIO CALCATREVERES
REGULCIPERIO CALCATR        20
NOMBRE (F=FIN): CESAR AUGUSTO
CESAR AUGUSTO               20
NOMBRE (F=FIN): PACO
PACO                        20
NOMBRE (F=FIN): P
P                           20
NOMBRE (F=FIN): F
Ok
```

Hemos adosado 20 espacios a NOMB$, aunque se vea poco. Daría igual haber añadido 40 espacios en blanco. O 222. Porque en la línea siguiente vamos a quedarnos sólo con los veinte primeros caracteres.

SPACE$(X)

Hay una función que nos libra de esas largas cadenas de «blancos» que muy a menudo se necesitan. Es la función:

```
SPACE$(N)
```

En la que N es un número entero comprendido entre 0 y 255. La cadena queda cargada con tantos « » (espacios en blanco) como sea el valor de esa N.

La línea 40 del programa anterior la podríamos escribir mejor, entonces:

```
40 NOMB$=NOMB$+SPACE$(20)
```

La filosofía del programa anterior es aplicable a multitud de casos semejantes. Imaginemos, por ejemplo, que queremos «rellenar» con ceros» por la izquierda, hasta completar a 7 cifras, referencias o códigos numéricos de longitud variable:

PR-1.28: rellenado de «0»s por la izquierda hasta completar a 7 cifras:

```
10 INPUT "Numero (0=FIN): ",N$
20 IF LEN(N$)>7 THEN 10
30 IF N$="0" THEN END
40 N$=RIGHT$("0000000"+N$,7)
50 PRINT N$:GOTO 10

RUN <CR>
Numero (0=Fin): 123
0000123
Numero (0=fin): 7654321
7654321
Numero (0=fin): 44
0000044
Numero (0=fin): 5454
0005454
Numero (0=fin): 2
0000002
Numero (0=Fin): 0
Ok
```

Si en lugar de rellenar con «0»s, quisiéramos rellenar con «blancos», sólo habría que cambiar la línea 40 (que es la responsable del meollo del trabajo):

```
40 N$=RIGHT$("0000000"+N$,7)
```

por

```
40 N$=RIGHT$(SPACE$(7)+N$,7)
```

con lo que conseguiríamos que los números quedaran perfectamente «justificados» por la derecha, como a todo número por derecho le corresponde.

Hemos visto algunas de las aplicaciones que la operación «suma de cadenas» (+) puede tener.

Y ya no existen otras operaciones «aritméticas» (en sentido figurado). No se pueden «restar» cadenas, por ejemplo.

En cambio, las cadenas alfanuméricas, sí admiten todas las operaciones relacionales así como las lógicas.

Veámoslo con ejemplos concretos:

PR-1.29: Comparación de cadenas.

```
10 INPUT "Palabra (F=Fin): ",A$
20 IF A$="F" THEN END
30 INPUT "Otra Palabra   :",B$
40 IF A$>B$ THEN PRINT A$;" ES MAYOR QUE ";B$
50 IF A$<B$ THEN PRINT A$;" ES MENOR QUE ";B$
60 IF A$=B$ THEN PRINT A$;" ES IGUAL A ";B$
70 PRINT:GOTO 10

RUN <CR>
Palabra (F=Fin): ANTONETE
Otra Palabra   : ZACARIAS
ANTONETE ES MENOR QUE ZACARIAS

Palabra (F=Fin): RICARDO
Otra Palabra   : RICARDA
RICARDO ES MAYOR QUE RICARDA

Palabra (F=Fin): AMOR
Otra Palabra   : AMOR
AMOR ES IGUAL A AMOR
```

```
Palabra (F=Fin): ODIO
Otra Palabra   : OPIO.
ODIO ES MENOR QUE OPIO.

Palabra (F=Fin): YO
Otra Palabra   : WENCESLAO
YO ES MAYOR QUE WENCESLAO

Palabra (F=Fin): CARLOS
Otra Palabra   : Carlos
CARLOS ES MENOR QUE Carlos

Palabra (F=Fin): ZUMALACARREGUI
Otra Palabra   : ZUNIA
ZUMALACARREGUI ES MENOR QUE ZUNIA

Palabra (F=Fin): ala
Otra Palabra   : ZUECO
ala ES MAYOR QUE ZUECO

Palabra (F=Fin): 99999999
Otra Palabra   : AARON
99999999 ES MENOR QUE AARON

Palabra (F=Fin): F
Ok
```

(Hemos hecho tan largo el ejemplo para que, meditando el lector sobre él, nos excuse de extendernos en explicaciones.)

Cualquier otro tipo de operador relacional o lógico funcionaría de un modo análogo. En todo lo que sigue de este texto surgirán numerosos casos sobre este tema.

OTRAS FUNCIONES DE CADENAS

STRING$(X,X$)

Así como hace un momento hemos visto, SPACE$ proporciona una cadena de espacios en blanco, existe una función que nos da la repetición de *cualquier* carácter. Es la:

```
STRING$(N,A$)
```

Significa (String = Cadena): «Cadena formada por N (de 0 a 255) caracteres iguales a A$». Si contiene más de un carácter, la función STRING$ toma por defecto *el primer* carácter (el primer byte, en general) de dicha variable.

Así pues, la asignación: KK$ = STRING$(7,«K») adjudica a la fea variable KK$ la cadena (aún más fea): «KKKKKKK».

Fácil es constatar la absoluta identidad entre:

```
SPACE$(80)    y    STRING$(80," ")
```

Enseguida volveremos sobre esta útil función.

STR$(N) VAL(X$)

Veamos ahora este par de funciones de muy frecuente uso, en cierto modo antinómicas.

Sabemos que 247 no es lo mismo que «247». 247 es una constante numérica que significa nada menos que «cien veces dos más diez veces cuatro más siete». Hay toda una codificación implícita en cualquier constante numérica. En cambio «247» no són más que tres caracteres adosados. Bien. Muy a menudo sucede que nos es preciso convertir valores numéricos (o variables id.) en cadenas y viceversa.

La función que lleva a cabo la primera transformación es:

STR$(N)

Cuya traducción al lenguaje llano es «cadena que corresponde al valor de N». (Ojo: no confundir STR$ con la STRING$ que acabamos de estudiar).

Por consiguiente, STR$(247), parece que tiene que ser igual que «247». Casi. En realidad, es igual a « 247» con un blanco en cabeza. Esto, que no deja de ser una lata, se debe a que en el número 247 hay un signo «+» delante, elíptico, fantasmal.

En cambio, STR$(-247) es igual a «-247», porque el signo «-» hay que ponerlo, ya no es elíptico.

(Recordamos que si A$ = «MARIA», MID$(A$,2) es igual a ARIA»).

En números positivos, para quitar ese molesto espacio inicial, no hay más que poner MID$(STR$(N),2).

Vamos a hacer un pequeño programa ejercicio que aclare ideas:

```
10 A=445:B=-643:C=66.87
20 A$=STR$(A):B$=STR$(B):C$=STR$(C)
30 PRINT A,B,C
40 PRINT A$,B$,C$
50 PRINT LEN(A$),LEN(B$),LEN(C$)
60 A1$=MID$(A$,2)
70 PRINT A1$,LEN(A1$)

RUN <CR>
 445          -643         66.87
 445          -643         66.87
 4             4           6
445            3
Ok
```

Moléstese el lector en explicar esas salidas de acuerdo con los párrafos anteriores.

Si hemos hecho esta asignación: A = 200∧4, y a continuación ponemos COSA$ = STR$(A) y luego preguntamos cuál es la LEN(COSA$), nos llevaríamos quizás una sorpresa. Si 200∧4 = 1600000000, STR$(A) se debe convertir en « 1600000000».

Pues no. A menos que A sea una variable numérica de doble precisión (hay una forma de conseguirlo, como ya veremos), lo que entra en COSA$ es el leve engendro: « 1.6E+08». Nos ha saltado la coma flotante porque el resultado de 200∧4 tiene más de 6 cifras significativas.

La función:

VAL(A$)

lleva a cabo la transformación inversa. Es decir, nos da el *posible* valor numérico del argumento de la función VAL. En este caso, de A$.

Por ejemplo, si A$ = «123456», B$ = «12.7» y C$ = «-9», los VAL(A$), VAL(B$) y VAL(C$) serán respectivamente 123456, 12.7 y -9.

Muy fácil. ¿Y si A$ = «INES», B$ = «12 TIOS», C$ = «14 millones» y D$ = «TRESCIENTOS»? Bueno. Por mucho que valga la dulce INES cosa que nadie pone en duda, su «valor numérico» es 0. Y lo mismo sucede con D$. B$ y C$ son las únicas cuyo VAL es capaz de recoger algo: 12 y 14, respectivamente.

Y si A$ = «188J443», VAL(A$) resulta ser igual a 188. En cuanto hay un carácter «no numérico», la función VAL, desesperada, abandona. Ya no busca más.

Atención con lo que *es* y con lo que *no es* valor numérico.

Si ponemos N = VAL(«1000E2») parece que, según lo anteior, N va a valer 1000, cuando en realidad lo que vale es 100000. La razón es que «E» (seguida de un número positivo o negativo) es un carácter numérico excepcional, equivale a «10 elevado al número que le sigue». Recuérdese lo que decíamos al tratar de la coma o el punto flotante. Así pues, N es 1000*10∧2 = 1000*100 = 100000.

En cambio N = VAL(«1000EA) dará 1000 para N.

INSTR(A$,B$)

La función *INSTR* es característica del Basic de Microsoft y aunque su aplicación no es excesivamente frecuente, tiene en ocasiones una muy fecunda utilidad.

Su forma general es:

```
INSTR(A,CADENA$,SUB$)
```

que se traduce por: «Posición a partir de la número A en que la subcadena SUB$ se encuentra en la cadena CADENA$». El parámetro A puede omitirse, siendo entonces 1 su valor por defecto. Si SUB$ no forma parte de CADENA$ entonces es nulo el valor de esta función («no lo encuentra»).

PR-1.30; Ejemplo de empleo de INSTR.

```
10 M$="JOSEJORGEIRENECOSMEMARIAJAIMEALICIAELENA"
20 P=INSTR(M$,"IRENE")
30 Q=INSTR(M$,"JO")
40 R=INSTR(2,M$,"JO")
50 S=INSTR(20,M$,"J")
60 T=INSTR(M$,"Z")
70 U=INSTR(M$,"MEMA")
75 V=INSTR(4,M$,"SE")
80 PRINT P;Q;R;S;T;U;V

RUN <CR>
 10  1  5  25  0  18  0
Ok
```

Analizando el ejemplo precedente, nos haremos cargo del funcionamiento de INSTR. Aconsejamos al lector ponga a prueba su diligencia y le saque punta al ejercicio anterior.

Y ahora, para seguir adelante, tenemos que profundizar un poco en las cadenas.

CODIFICACION ASCII

Sabemos que un «BYTE» es el conjunto de 8 bits. Los caracteres que hasta ahora venimos manejando, los «eslabones» de las cadenas alfanuméricas no son sino bytes, «bytes visibles», llamémoslos así puesto que son representables por un carácter, o signo visible. Hay caracteres numéricos, alfabéticos y especiales.

(Dejamos aparte que los bytes, además, pueden ser instrucciones directas al microprocesador. Ello sucede cuando trabajamos, no en BASIC, sino en algún lenguaje de muy bajo nivel, así que, no se alarme el lector y siéntase libre de semejante horror. Ya le llegará el día, si mantiene su interés por la programación, de meterse en esos berenjenales.)

Dentro del BASIC que nos ocupa, y dentro del concepto de byte como «unidad encadenable» (como «carácter», en un sentido muy general), hay bytes que no tienen representación en forma de signo visible.

Uno de ellos lo estamos empleando ya con profusión y es el CR. Otro es el LF —Line Feed— o salto de línea (hacia abajo). Otro, el CS —clear screen—, borra la pantalla. El BS —back space— provoca el retroceso del cursor, etc.

Hagamos una informal clasificación de nuestros bytes u octetos:

```
CARACTERES VISIBLES:      ALFANUMERICOS: ALFABETICOS: A  B  Z
                                         NUMERICOS: 1  2  0  9

                          ESPECIALES: (  ?  )  #  &

NO VISIBLES (NO SIGNOS): DIRECTOS: P.ej. 'LF','CR','FF'

                          DE TRANSFORMACION: P.ej. 'ESC'
```

Dentro de la relativa informalidad de esta clasificación digamos que en la práctica a todos ellos se los denomina a veces, como «ca-

racteres» a secas y que se suele incluir dentro de la denominación de «alfanuméricos», también a los «especiales». Y aún, en ocasiones se hace referencia a «cadenas alfanuméricas» en las que pueden estar incluidos hasta caracteres no visibles.

De los bytes que llamamos no visibles, damos el nombre de «directos» a los que su aplicación produce un efecto inmediato sin auxilio de nadie: por ejemplo, si en LF$ hemos cargado el byte que corresponde a «salto de línea», la instrucción:

```
PRINT LF$;
```

producirá inmediatamente un salto a la línea siguiente, a pesar del punto y coma que le acompaña.

Y

```
PRINT LF$
```

así, sin punto y coma, producirá dos saltos de línea: uno debido al LF$ y otro al PRINT, como siempre.

Los bytes de transformación son aquellos que *transforman* el significado del (o de los) bytes que le siguen. Son bytes-llave.

(Esto no es nada misterioso. Siempre en nuestro afán desmitificador, el símil de estos bytes sería, en nuestra vida diaria, la «actitud» del que habla o su «entorno» o su «contexto». Piénsese cuán diferentes pueden ser los significados de palabras tan corrientes como «rico», «coger», «libre», «cubierto», o «leche», con perdón. Ni con la informat ica hay nada nuevo bajo el sol.

Por ejemplo si metemos en la variable ESC$ uno de estos potentes bytes, la instrucción:

```
PRINT ESC$;"="  (equivalente a PRINT ESC$+"=")
```

Producirá posiblemente cualquier cosa, excepto la aparición de un signo de igualdad.

No hay —no puede haber— más que 256 bytes diferentes, puesto que ésas son las posibles formas de agrupar «unos» y «ceros» (bits) tomándolos de ocho en ocho.

La información que se alberga en un ordenador, sea en su memoria interna o en su memoria externa (soporte magnético, disco, cassette, etc.) lo va a hacer *siempre* (no puede hacerlo de otra manera) en forma de bits, y éstos, agrupados en conjuntos de 8 constituyendo bytes u octetos.

Los Códigos ASCII han venido a proporcionar una correspondencia entre lo que es cada byte para la máquina y lo que es cada byte para el que tiene que programarla o utilizarla (programador o usuario). Y esa correspondencia ha sido aceptada casi universalmente.

De la misma manera que de Gibraltar a Helsinki se entiende lo que es «7» sin necesidad de más explicaciones (aunque se le pronuncie «seven» o «sept» o «sieben» o «siete»), lo que para un ordenador japonés es el byte «01000001» (65, traducido a decimal), lo es también para uno fabricado en el famoso Silicon Valley de California.

Como el número de letras, guarismos, signos especiales, etc., no llega a cubrir las 256 posibilidades que el byte ofrece, la codificación ASCII sólo «obliga» a un cierto número de esos posibles bytes, dejando amplias zonas del espectro a la libre disposición de quien diseña el ordenador.

En el último Apéndice figura la tabla completa de Códigos ASCII, para cualquier consulta del lector.

El primer carácter «visible» (y muy poco, por cierto) se corresponde con el byte «00100000» (32 en decimal). Ese carácter es el «blanco» o «hueco» o «espacio». Es, por ejemplo, el carácter que separa dos palabras cualesquiera de lo que Vd. está leyendo ahora mismo. Los anteriores, del 0 al 31 (o en binario: del «00000000» al «00011111») corresponden a caracteres no visibles. Y del 32 inclusive en adelante, se encuentran todos los visibles.

Entre los invisibles mencionamos:

— LF que se corresponde con el byte 10

— CR con el 13

— TAB (tabulación) con el 9

En lo que se refiere al manejo de cadenas alfanuméricas, debemos dejar bien claro que una cadena puede estar formada por caracteres *de cualquier tipo* sin más restricción que su longitud (que no puede sobrepasar los 255 caracteres).

CHR$(X) Y ASC(X$)

Estamos ya en condiciones de meternos en el estudio de estas importantes funciones.

La función CHR$(Expresión Numérica) proporciona el «carác-

ter» cuyo código ASCII es el valor numérico del argumento que figura entre paréntesis. Lo que hemos llamado «Expresión Numérica» para generalizar.

Así pues, CHR$(65) (léase «carácter cuyo código ASCII es el 65») se corresponde exactamente con la letra «A» cuyo código ASCII es precisamente 65. Análogamente CHR$(66) es idéntico a «B» y así sucesiva y ordenadamente hasta la «Z», que se identifica con CHR$(90). Recordamos que el alfabeto *normal* en informática es el inglés, en el que no figura la «eñe» y que consta por tanto de 26 letras. Cuando se quiere expresar esa «eñe», se echa mano de un carácter especial (normalmente « \ ») que se sustituye por nuestra aislada letra.

Veamos el resultado de la ejecución de este pequeño programa:

```
10 CODI=65
20 PRINT CHR$(CODI);
30 CODI=CODI+1
40 IF CODI>90 THEN END
50 GOTO 20

RUN <CR>
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
```

Y ahora de este otro:

```
10 CODI=48
20 PRINT CHR$(CODI);
30 CODI=CODI+1
40 IF CODI>57 THEN END
50 GOTO 20

RUN <CR>
0123456789
Ok
```

Inversamente, la función que da el código ASCII de un carácter determinado es la *ASC* cuyo argumento en este caso es el carácter en cuestión.

Por ejemplo, si escribimos:

```
PRINT ASC("A")
```

(Léase «enséñame el código ASCII que corresponde al carácter «A») obtendremos «65» como resultado inmediato.

Y si pusiéramos:

```
PRINT ASC("ALEJANDRO")
```

conseguiríamos idéntico resultado. Porque la función ASC haría caso omiso de los caracteres que pudieran seguir al primero, único que tomaría en consideración. Algo parecido a lo que sucedía con la función STRING que sólo hacía caso a la primera letra.

Hagamos ejecutar el siguiente miniprograma, inverso del anterior:

```
10 A$="0123456789"
20 SITUA=1
30 PRINT ASC(MID$(A$,SITUA,1));
40 SITUA=SITUA+1
50 IF SITUA>10 THEN END
60 GOTO 30

RUN <CR>
 48  49  50  51  52  53  54  55  56  57
Ok
```

Si se nos ocurriera escribir: PRINT ASC(CHR$(65)) obtendríamos ipso facto «65». Y análogamente CHR$(ASC(«A»)) sería igual a «A», con lo que habríamos hecho un par de panes con un par de tortas.

Probemos a ejecutar ahora la siguiente instrucción, en modo directo:

```
PRINT STRING$(20,CHR$(66)) <CR>
```

nos saldrá

```
BBBBBBBBBBBBBBBBBBBB
Ok
```

Y aún nuestro Basic nos permite escribir:

```
PRINT STRING$(20,66)
```

con un resultado exactamente igual. Es decir, en la función STRING$, está permitido sustituir el segundo parámetro, el que define el carácter a repetir, por su código ASCII.

Así, el STRING$(250,0) será una cadena de 250 «caracteres» todos ellos constituidos por bytes «nulos» (en binario «00000000»).

Si escribimos: PRINT LEN(STRING$(250,0)) obtendremos un «250» que significa que por muy nula que sea la cadena formada

(consta nada menos que de 250*8 = 2000 ceros o 2000 bit-ceros) está ocupando un lugar en la memoria.

Probemos ahora:

```
10 A$="ERNESTO"+STRING$(200,0)+"CATALA"
20 PRINT A$, LEN(A$)

RUN <DR>
ERNESTOCATALA      213
Ok
```

La parte de A$ formada por los 200 caracteres nulos, no producen ningún efecto en la pantalla, y sin embargo ocupan lugar dentro de la memoria interna, (o externa) como lo demuestra su LEN.

(Esta no aparición en pantalla de —por ejemplo— los CHR$(0) no es general. Algunos ordenadores lo traducen a pantalla por un espacio y aún por otras rarezas.)

A continuación, vamos a hacer un programa en toda regla.

Dos amigos, hombres de letras, se encuentran discutiendo acerca de la frecuencia con que, en un determinado idioma (por ejemplo, el castellano) aparecen, estadísticamente las letras en cualquier texto literario.

Se trata de hacer un programa, lo más corto posible, que sea capaz de analizar un texto cualquiera y sacar el número de veces que aparece cada una de las 26 letras (prescindimos de «ñ»).

Suponemos que hasta la línea 400 del programa se han cargado en un vector alfanumérico 200 líneas de texto mediante lectura de un disco cuyo procedimiento no corresponde aquí (véase dentro de esta colección «Ficheros Basic»). Entramos por lo tanto en la línea 400 con el vector LINEA$(1), LINEA$(2).... LINEA$(200) ya en condiciones de ser analizado.

PR-1.31: Análisis de los caracteres de un texto, en el que todo está escrito con mayúsculas.

```
.....

400 LIN=1:COL=1
410 COD=ASC(MID$(LINEA$(LIN),COL,1))-64
420 IF COD>0 AND COD<27 THEN N(COD)=N(COD)+1
440 COL=COL+1
450 IF COL<=LEN(LINEA$(LIN)) THEN 410
480 LIN=LIN+1
```

```
490 IF LIN<=MAXLIN THEN COL=1:GOTO 410
520 PRINT "CARACTER    VECES"
530 PRINT "--------    -----"
540 I=1
550 PRINT "   ";CHR$(I+64);"        ";N(I)
560 I=I+1:IF I>26 THEN END
570 GOTO 550
```

(Cuando estudiemos el ciclo FOR - NEXT veremos que esto aún se puede simplificar).

En el programa anterior hemos utilizado las variables LIN y COL (Línea y Columna) para rastrear con ellas todo el escrito, habida cuenta de que el número de columnas (longitud de línea) de cada línea puede ser variable y de que son 200 líneas.

En la variable COD queda cargado en la 410 el código ASCII de cada letra. La 420 sirve para filtrar (desechando) todos los caracteres que encuentra que no sean letras (espacios, puntos, etc.). En la misma 420 van incrementándose sucesivamente los «contadores de letras» que son los elementos del vector N. Por ejemplo, si la letra encontrada es una «D», COD será igual a 4 según la 410, y N(4) aumentará una unidad. Y así todos.

De la línea 500 en adelante se produce la visualización por pantalla de los resultados que dirimirá la discusión entre los dos amigos.

CAPITULO 6.— FUNCIONES PARA TODO

FUNCIONES NUMERICAS

Toda función se compone de unas cuantas letras (algunas veces con pretensión de «palabra») y de un argumento o «cosa» sobre la que va a actuar la función para sacarle algún jugo. En muy pocas ocasiones este argumento puede omitirse por innecesario.

Hemos visto ya algunas funciones. Por ejemplo, LEN(A$), es de suyo una función numérica (da un número) aunque su argumento sea una variable alfanumérica. A la función STR$(A), le ocurre lo contrario: es una función alfanumérica cuyo argumento es un número o una variable numérica.

Veamos ahora unas cuantas funciones numéricas cuyo argumento es también numérico, todas ellas muy sencillas.

INT

Da la *parte entera* del argumento. Con más propiedad: parte entera del argumento, tomado por defecto (aunque parezca una redundancia).

Ejemplos:

```
PRINT INT(2.3); INT(.9); INT(12/5) <CR>
 2  0  2
Ok
```

Como acontece siempre con las funciones, el argumento puede ser una constante, una variable o una expresión numérica sin límite en la práctica en cuanto a su complicación aritmética o lógica.

Por ejemplo:

```
A=3:B=4:C=INT((A/B)^A-A*B/7+11.23)
```

Si el argumento es negativo, y con más propiedad de lo que al pronto parece, la función INT redondea al número negativo *inferior.* (Por eso no era redundante la definición, al añadir «por defecto»). Entonces:

```
PRINT INT(-123.87)
-124
Ok
```

FIX

Muy parecida a la anterior. Es como el INT que uno se imagina a primera vista, es decir, que esta función se limita a despreciar los decimales cualesquiera que éstos sean y cualquiera que fuere el número (positivo o negativo).

```
PRINT FIX(-123.87);FIX(3/4);FIX(100/3);FIX(-7/3) <CR>
-123  0  33 -2
Ok
```

ABS

Da el valor absoluto de un número o de una expresión numérica, o sea que convierte en positivo lo negativo.

```
PRINT ABS(12);ABS(-45);ABS(INT(-13/6)) <CR>
 12  45  3
Ok
```

SQR

(Caricatura de «square root», raíz cuadrada) da la raiz cuadrada del argumento:

```
PRINT SQR(625);SQR(1/16);SQR(-4) <CR>
 25  .25
Illegal function call
Ok
```

(No le gusta al Basic eso de los números imaginarios, y recha za el SQR(-4).)

Preguntamos: ¿hay alguna diferencia entre:

```
SQR(345)  ,  345^(1/2)  y  345^.5  ?
```

No. La única pequeña diferencia está en el «modo» en que el Basic obtiene los resultados, lo cual de momento nos trae sin cuidado.

SGN

(Apócope de «sign» o signo). Sólo puede tomar tres valores: 1 si el argumento es positivo, -1 si es negativo y 0 si es nulo.

```
PRINT SGN(-876543),SGN(INT(.8)),SGN(1/1000) <CR>
-1              0               1
Ok
```

¿Y cuál será el valor de SGN(PEPE)? Pues todo dependerá del valor que tenga la variable PEPE. Seguro que -1, 0 ó 1, sin remisión.

FRE

(De «free», libre). Es una función especial que, utiliza un argumento simbólico, del que no hace ningún caso, pues se limita a proporcionar el número de bytes que quedan «libres» en memoria interna, a disposición del programador.

```
PRINT FRE("") <CR>
 22465
Ok
```

Que quiere decir que *en ese momento*, el programador cuenta con 22645 bytes a su disposición, para datos y programa.

Aparte hay que echar de comer a la función que a continuación estudiamos.

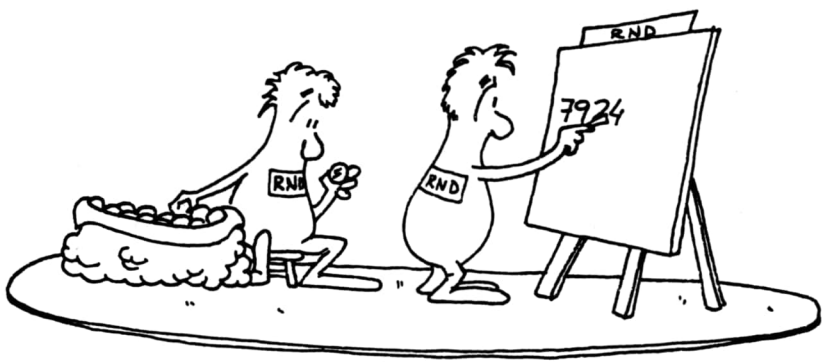
RND

(Contracción de «Random = aleatorio, al azar). Esta curiosa y con frecuencia, muy útil función (que tampoco hace caso de lo que tenga el argumento hasta el punto que se puede suprimir éste), genera un número comprendido entre 0 y 1, con seis cifras decimales, y con toda la apariencia de haber sido buscado al azar, aunque realmente no lo sea (se denominan por eso, números pseudoaleatorios, los que la función RND obtiene). Cada vez que el programa pasa por esta función, el intérprete parece sacarse de la manga un nuevo número aleatorio, sin ninguna relación con el anterior.

```
10 N=RND
20 IF N>.85 THEN END
30 PRINT N
40 GOTO 10
```

```
RUN <CR>
 .245132
 .775004
 .691332
 .422901
 .212854
 .2.33421E-03
Ok
```

Esto parece carecer de todo sentido práctico. Bueno, por ahora sí. Pero en el trabajo de un programador hay múltiples ocasiones en las que tiene que simular algo: una ordenación, una grabación, el funcionamiento de una red de distribución de agua, los costes de producción de una fábrica... Y es bueno poder echar mano de unos datos inventados rápidamente por «alguien». Ese alguien es nuestro ordenador cuya velocidad de generación de números aleatorios es considerable (más de 200 por segundo).



En la práctica, los números aleatorios que necesitamos están comprendidos en un intervalo distinto al 0-1 que nos proporciona la máquina. Pero es fácil la transposición de intervalos.

Por ejemplo, vamos a ver cómo convertiremos el RND en el número que sale tirando un dado sobre una mesa. (Téngase en cuenta que el RND nunca produce el 0 ni el 1 redondos, se trata de lo que los matemáticos llaman un intervalo abierto).

El valor que toma N en la asignación:

```
N=INT(RND*6)+1
```

resuelve nuestro problema, porque indefectiblemente N tendrá un valor entero comprendido entre 1 y 6.

Y si se trata de sacar un número de ruleta al azar, la asignación será:

```
N=INT(RND*37)
```

donde el valor que tomará N será entero y comprendido entre 0 y 36.

Y por último:

```
N=INT(RND*2)
```

dará, aleatoriamente, 0 ó 1 lo que da respuesta a cualquier problema de tipo dicotómico (moneda al aire, suertes simples de ruleta, generación de bits, etc.)

PR-1.32: Juego de dados que demuestra el cumplimiento de las leyes de los grandes números. Se tira 6000 veces un dado y se trata de saber cuántas veces ha salido cada número.

```
10 NUMERO=INT(RND*6)+1
20 VECES(NUMERO)=VECES(NUMERO)+1
30 TOTAL=TOTAL+1
40 IF TOTAL>6000 THEN 100
50 GOTO 10
100 DADO=1
110 PRINT "El Numero";DADO;"ha salido";VECES(DADO);"veces."
120 DADO=DADO+1:IF DADO>6 THEN END
130 GOTO 110

RUN <CR>
El Numero 1 ha salido 984 veces.
El Numero 2 ha salido 1040 veces.
El Numero 3 ha salido 1008 veces.
El Numero 4 ha salido 975 veces.
El Numero 5 ha salido 1033 veces.
El numero 6 ha salido 960 veces.
Ok
```

PR-1.33: Y ahora juguemos a la ruleta. Estamos en Montecarlo o en La Manga. Al principio se nos pregunta la cantidad total que estamos dispuestos a jugarnos (o sea, a perder). En cada jugada elegimos número al que vamos a jugar y el importe de la apuesta.

```
20 INPUT "Cuanto dinero vas a jugarte en total? ",RIESGO
40 SALDO=RIESGO
50 PRINT
60 INPUT "A que numero apuestas? ",SUERTE
65 IF SUERTE<0 OR SUERTE>36 OR INT(SUERTE)<>SUERTE THEN 60
70 INPUT "Cantidad que te juegas? ",CANTI
90 IF CANTI<200 THEN PRINT "Apuesta minima 200.":GOTO 70
100 IF CANTI>SALDO THEN PRINT "Solo te queda ";SALDO:GOTO 70
110 NUMERO=INT(RND*37)
120 PRINT
140 PRINT "Ha salido el";NUMERO
160 IF NUMERO=SUERTE THEN 300
180 SALDO=SALDO-CANTI
190 PRINT:PRINT "Te quedan ";SALDO;" Pts."
200 IF SALDO<200 THEN GOTO 250 ELSE PRINT:GOTO 60
250 PRINT "No puedes seguir jugando. Hasta otra.":END
300 PRINT: GANAS=35*CANTI
310 PRINT "Has tenido suerte! Has ganado";GANAS;" Pts"
315 SALDO=SALDO+GANAS
320 PRINT "Tu saldo asciende a";SALDO
330 INPUT "Si no quieres seguir jugando pulsa N ",J$
350 IF J$<>"N" AND J$<>"n" THEN PRINT:GOTO 60
360 PRINT:GANANCIA=SALDO-RIESGO:G=ABS(GANANCIA)
370 IF GANANCIA>0 THEN PRINT "Has ganado en total";G;" Pts.":END
380 IF GANANCIA=0 THEN PRINT "Te has divertido gratis.":END
390 PRINT "Has perdido";G;" Pts. Otra vez tendras mas suerte."
```

PR-1.34: Ahora vamos a jugar a un número fijo y una cantidad fija cada vez. El ordenador jugará automáticamente hasta conseguir ganar una cantidad prefijada (u, horror, perderlo todo). Se advierte que si se pretende ganar mucho y el riesgo es grande, es muy posible que pasen horas hasta llegar a un final.

```
10 PRINT
20 INPUT "Dinero total que quieres jugarte (0=FIN): ",RIESGO#
30 IF RIESGO#=0 THEN END ELSE SALDO#=RIESGO#
40 JUG=0
50 INPUT "Ganancias con las que te retirarias: ",GANANCIA#
60 INPUT "Numero al que vas a apostar siempre: ",SUERTE
62 IF SUERTE<0 OR SUERTE>36 OR SUERTE<>INT(SUERTE) THEN 60
70 INPUT "Cantidad fija que apuestas: ",CANTI
```

```
80 IF CANTI<200 THEN 70
90 PRINT
100 NUMERO=INT(RND*37)
102 PRINT NUMERO;
110 JUG=JUG+1
120 IF NUMERO=SUERTE THEN 160
130 SALDO#=SALDO#-CANTI
140 IF SALDO#<200 THEN PRINT "Has perdido";RIESGO#-SALDO#:GOTO 190
150 GOTO 100
160 SALDO#=SALDO#+CANTI*35
170 IF SALDO#-RIESGO# <GANANCIA# THEN 100
180 PRINT "Has ganado";SALDO#-RIESGO#
190 PRINT:PRINT "En un total de ";JUG;"jugadas.":PRINT:GOTO 10
```

Creemos que el lector dispone, con estos programas, de ideas suficientes como para organizar él mismo su propia estrategia de juego.

Y regresamos a nuestras funciones numéricas.

Las trigonométricas (SIN, COS, TAN, ATN), las logarítimicas (LOG, EXP) y las importantísimas funciones PEEK e INP las dejamos para la Segunda Parte. Las funciones de pantalla —TAB y POS— las incluimos en capítulo posterior, dedicado todo él al manejo de este periférico desde el Basic.

OTRAS FUNCIONES DE CADENA

En el capítulo anterior hemos analizado con detenimiento la mayor parte de las funciones alfanuméricas (LEFT$, MID$, etc.). Quedan por estudiar dos que, por su naturaleza, hemos preferido dejar aparte y analizar meticulosamente. Son las funciones de entrada por teclado *INPUT$* e INKEY$.

INPUT$

Actúa de modo semejante a la *sentencia* INPUT (con la que no debe confundirse). Cuando el programa se encuentra con esta función, queda esperando a que en el teclado sean pulsados tantos caracteres como indique el argumento de INPUT$ que siempre será un número o una expresión numérica. A diferencia con el INPUT normal:

— Las teclas que se pulsen no aparecen (no hacen *eco*) en pantalla.

— Cuando se pulsa la última (la que completa el número indicado en el argumento), el programa continúa en su siguiente instrucción *sin esperar* que sea apretado el CR.

La aplicación de esta función es de una riqueza insospechada.

PR-1.35: Programa (subprograma) de identificación de un usuario del ordenador.

```
1000 PRINT "IDENTIFIQUESE TECLEANDO SU CLAVE"
1010 Y$=INPUT$(10)
1020 IF Y$="AGENTE 007" THEN 3000
1030 GOTO 1000
3000 PRINT "De acuerdo, Vd. es el ";Y$
3010 PRINT "El programa puede seguir."
3020 PRINT "Bla bla bla (continuacion del programa..)"

RUN <CR>
IDENTIFIQUESE TECLEANDO SU CLAVE
IDENTIFIQUESE TECLEANDO SU CLAVE
IDENTIFIQUESE TECLEANDO SU CLAVE
De acuerdo, Vd. es el AGENTE 007
El programa puede seguir.
Bla bla bla (continuacion del programa..)
Ok
```

Al tercer intento, el usuario ha sabido teclear la clave correcta. (Seguramente en los dos primeros intentos ha utilizado minúsculas. O simplemente ha apretado una tecla errónea).

Fácilmente se comprende que si el argumento del INPUT$ es «1» (o una expresión numérica que valga 1), el programa se parará, a la espera de que se toque *una sola* tecla que, sin aparecer en pantalla, hará que el programa continúe.

El hecho de que lo que se teclee no aparezca en pantalla no significa que la variable que recoge la función no quede correctamente cargada con el —o los— caracteres tecleados.

PR-1.36: Salida de tabla por pantalla con scrolling interrumpido para posible consulta de cada «pantallazo».

```
10 PRINT "Numero     Cuadrado       Cubo"
20 PRINT "-----------------------------"
30 N=1
40 PRINT N,N^2,N^3
50 N=N+1
60 IF N/20=INT(N/20) THEN Y$=INPUT$(1):IF Y$="F" THEN END
70 GOTO 40
```

La salida de la tabla se interrumpe de 20 en 20 conforme a lo que la línea 60 ordena. Se ve entonces al cursor en actitud de espera. Basta tocar cualquier tecla para que la salida de datos prosiga. Algunos datos salen con unos extraños apéndices decimales, a todas luces erróneos. No, no está mal el programa. Son gracias del Sistema, que cuando utiliza la potenciación (↑) comete frívolamente ligeros errores debido al procedimiento que la unidad aritmética emplea en el cálculo. No se preocupe. Basta con cambiar «N↑2» por su equivalente «N*N» y «N↑3» por «N*N*N» para que las cifras ya salgan bonitas.

(Mejor quedaría el programa anterior, utilizando variables de doble precisión).

PR-1.37: El programa anterior, mejor acondicionado.

```
10 PRINT
20 PRINT"TABLA DE CUADRADOS Y CUBOS":PRINT
30 INPUT "Desde que numero empezamos (0=Fin) ";A#
40 IF A#=0 THEN END
50 PRINT "Numero     Cuadrado         Cubo"
60 PRINT "-------------------------------"
70 N#=A#
80 PRINT N#,N#↑N#,N#↑N#↑N#
90 N#=N#+1
100 IF N#/20=INT(N#/20) THEN Y$=INPUT$(1):IF Y$="F" THEN PRINT:GOTO 30
110 GOTO 80
```

Y ya puestos a buscar estética, vamos a hacer, anticipándonos a lo que veremos dos capítulos más adelante, que las tres columnas de cifras, queden justificadas por la derecha en lugar de estarlo por la izquierda.

PR-1.38: El programa anterior con justificación por la derecha de los números.

```
10 PRINT
20 PRINT"TABLA DE CUADRADOS Y CUBOS":PRINT
30 INPUT "Desde que numero empezamos (0=Fin) ";A#
40 IF A#=0 THEN END
50 PRINT "Numero     Cuadrado         Cubo"
60 PRINT "-------------------------------"
70 N#=A#
74 N$=RIGHT$(SPACE$(6)+STR$(N#),6)
76 NN$=RIGHT$(SPACE$(12)+STR$(N#↑N#),12)
```

```
78 NNN$=RIGHT$(SPACE$(12)+STR$(N#)+N#+N#),12)
80 PRINT N$;NN$;NNN$
90 N#=N#+1
100 IF N#/20=INT(N#/20) THEN Y$=INPUT$(1):IF Y$="F" THEN PRINT:GOTO 30
110 GOTO 74
```

INKEY$

Es una función que carece de argumento. Se semeja mucho al INPUT$(1) utilizado en los programas anteriores. La diferencia es que así como el INPUT$(1) *espera* la pulsación de una tecla, el INKEY$ *no espera.* De ese modo, la «no pulsación» de tecla mantiene a esta función con la «cadena nula» («») que ya conocemos. Hasta el momento en que se pulse cualquier tecla, en cuyo momento esta función queda cargada con el carácter que corresponde a la tecla pulsada.

Si en el último programa nos limitamos a cambiar la línea 100 por esta:

```
100 Y$=INKEY$:IF Y$="" THEN 100 ELSE IF Y$="F" THEN 30
```

veamos lo que pasa. Si no se toca el teclado, la función INKEY$ (y por consiguiente Y$) será igual a «''''» con lo que el programa quedará interrumpido en esa línea. En cuanto toquemos una tecla, Y$ se cargará con el carácter pulsado. Si ese carácter es la "F" saltará a la 30 y si es otro carácter cualquiera pasará a la línea siguiente, con lo que aparecerá en pantalla una línea de datos. Es decir, pulsando por ejemplo la barra de espacios, provocaremos —con cada toque— la salida de una línea en pantalla.

Una estructura de instrucciones parecida a la anterior línea 100 es la típica que se emplea para provocar interrupciones en todos los trabajos que conllevan salidas masivas de información por pantalla o impresora.

```
1190 W$=INKEY$:IF W$=CHR$(27) THEN 220
```

Esta línea 1190 suponemos que está inmersa en plena zona de programa donde se dan las instrucciones de salida de datos por un periférico de salida (pongamos, una impresora). Y suponemos asimismo que en la 220 hemos escapado de esa zona del programa. Pues bien, si no tocamos ninguna tecla, el programa ignorará a esa 1190. Más aún, si tocamos cualquier tecla que no sea la que corresponde al CHR$(27) (que es el carácter de Escape, tecla «ESC» o "BREAK") tampoco sucederá nada, el periférico seguirá obedeciendo

a las instrucciones de salida (seguirá "escribiendo"). Sólo cuando pulsemos el "ESC" (cuyo ASCII corresponde al 27), el programa bifurcará a la 100.

RESUMEN DE LAS FUNCIONES DEL BASIC

Con lo que hemos visto en el capítulo anterior y en lo que va de este, existe un repertorio de funciones que el BASIC tiene preparadas, que ya están disponibles para su uso inmediato en los programas.

El esquema general de su empleo es:

VARIABLE. FINAL = FUNCION (VARIABLE. INICIAL)

Con ello expresamos que, dada la variable VARIABLE. INICIAL y la función FUNCION, en la VARIABLE. FINAL queda cargado el resultado de la asignación. Según la naturaleza de VARIABLE. INICIAL y de FUNCION, se pueden dar las cuatro siguientes variantes que resumimos en forma de cuadro:

VAR.INICIAL	FUNCION	VAR.FINAL	E j e m p l o s
Numèrica	Numèrica	Numèrica	A = INT(B): X = ABS(Y)
Numèrica	Alfanum.	Alfanum.	A$ = STR$(A):J$=SPACE$(H)
Alfanum.	Numèrica	Numèrica	L=LEN(A$):E=VAL(A$)
Alfanum.	Alfanum.	Alfanum.	I$=LEFT$(A$,L)

Constatamos que la naturaleza (numérica o alfanumérica) de la variable final coincide con la de la función.

Parece al pronto que el número de funciones que nos ofrece el BASIC (las que hemos visto y algunas otras de menos frecuente uso que se explican en la segunda parte —BASIC SISTEMATICO—) es limitado. Pues no. Disponemos de un número *ilimitado* de funciones.

FUNCIONES DEFINIDAS POR EL PROGRAMADOR

Entramos como en otra dimensión. Disponemos, para programar, de todas las funciones que queramos sin otra limitación que la capacidad de memoria y, cómo no, el sentido común. Porque nuestro BASIC tiene prevista la poderosa sentencia *DEF FN* con la que definimos una función acomodada a nuestras necesidades perso-

nales. Digamos que todas las funciones que hemos estudiado, son prendas "pret-a-porter" y lo que ahora vamos a ver son auténticos trajes a medida.

Para mejor entrar en materia, situémonos en un supuesto caso práctico. Resulta que estamos confeccionando un programa en el que frecuentemente tenemos que aplicar una fórmula geométrica, por ejemplo, la superficie total de un cilindro de base circular.

Las dos variables geométricas que definen esa superficie son:

R: radio de la base y
H: altura del cilindro.

La formulita que daba esa superficie (quitemos las telarañas que pueda haber en nuestra memoria) era, puesto ya en formato BASIC con algunos espacios inoperantes para más claridad:

```
SUP = 2 * PI * R * (R + H)
```

En ella, la variable PI contiene el famoso 3.1416, mientras R y H son las variables antes definidas.

Si, como deciamos en nuestro supuesto, hemos de emplear esa fórmula varias veces en nuestro programa, echamos de menos la existencia de la función CILINDRO(R,H) que automáticamente nos diese el valor de la fórmula anterior.

Podemos hacer realidad nuestros deseos si mas que incluir, al principio del programa la siguiente instrucción:

```
10 DEF FNCILIN(R,H)=2*3.1416*R*(R+H)
```

Con esa instrucción acabamos de fabricar una función que permanecerá vigente a lo largo de todo el programa. Si, por ejemplo, más adelante aparece la asignación:

```
900 SUP=FNCILIN(17,13)
```

en la variable SUP quedará cargado el valor de la superficie total del cilindro cuyo radio es 17 y su altura vale 13. El único lastre que tenemos que soportar son las dos letras *FN* que acompañan a la función *CILIN* necesarias para que el sistema se entere que estamos refiriéndonos a una función definida por nosotros en el programa. Es indudable la ventaja de trabajar con estas funciones así definidas: nos ahorramos el trabajo de escribir fórmulas cuya complicación puede a veces ser grande, con la consiguiente redución de posibles errores mecanográficos.

Hemos definido (nos hemos "fabricado") una función numérica cuyo argumento (R y H en el ejemplo) es (son) también numéricos. Pero veamos otros ejemplos prácticos en los que jugaremos con la naturaleza de las funciones poniéndose de manifiesto la fecundidad de nuestras funciones.

Analicemos la función definida en la línea 12 (obsérvese que, a diferencia con la del cilindro, se trata de una función alfanumérica):

```
12 DEF FNNUM$(N)=RIGHT$(SPACE$(10)+STR$(N),10)
```

Con esta función conseguimos convertir al número (el que sea) N en una variable alfanumérica astutamente justificada a la derecha. Completemos con las sencillas instrucciones (recapacítese en la línea 110):

```
100 FOR I=1 TO 4
110 PRINT FNNUM$(20^I)
120 NEXT I

RUN <CR>
        20
       400
      8000
    160000
Ok
```

Y si el programa hubiera sido este:

```
20 DEF FNN$(N$,M)=RIGHT$(STRING$(30,"*")+STR$(N$),M)
30 '
90 X$=1
100 FOR I=1 TO 10
110 X$=-X$*20
120 PRINT FNN$(X$,25)
130 NEXT I

RUN <CR>
**********************-20
********************** 400
**********************-8000
********************** 160000
**********************-3200000
```

```
################ 64000000
###############-1280000000
############## 25600000000
#############-512000000000
########## 10240000000000
Ok
```

Veamos esta otra definición de función:

```
10 DEF FNL(A$)=ASC(RIGHT$(A$,1))
```

en ella incorporamos a la función L el valor del código ASCII del último carácter de la cadena A$.

El programa sigue:

```
20 INPUT X$:IF X$="0" THEN END
30 PRINT FNL(X$):GOTO 20

RUN <CR>
?ANTON
 79
?PEREZ
 90
?ANA
 65
?0
Ok
```

Volviendo al primer ejemplo, podríamos haber hecho la siguiente definición:

```
10 DEF FNC$(R,H)="Superficie del Cilindro ="+STR$(6.2832*R*(R+H))
```

Completando con:

```
20 PRINT
25 INPUT "Dame Radio y Altura: ",A,B
30 IF A=0 THEN END
40 PRINT FNC$(A,B):PRINT
50 GOTO 20
```

```
RUN <CR>
Dame Radio y Altura: 1,1
Superficie del Cilindro = 12.5664

Dame Radio y Altura: 13,28
Superficie del Cilindro = 3348.95

Dame Radio y Altura: 1.32,7.81
Superficie del Cilindro = 75.7226

Dame Radio y Altura: 0,0
Ok
```

En este ejemplo, como en otras ocasiones anteriores, hemos utilizado variables distintas en la *definición* y en su *utilización* en el programa. Con ello queremos resaltar que las variables utilizadas en la definición son ficticias, es decir, no quedan "secuestradas" para el resto del programa. Han sido empleadas por el BASIC con fines referenciales a la hora de definir la función, nada más.

Profundizando en el último ejemplo, podríamos incluso haber puesto:

```
10  DEF FNC$(R,H)="Superficie del Cilindro de radio"+
    STR$(R)+" y altura"+STR$(H)+" ="+STR$(6.2832*R*(R+H))
20  INPUT "R=",X
30  IF X=0 THEN END
40  INPUT "H=",Y
50  PRINT:PRINT FNC$(X,Y):GOTO 20

RUN <CR>
R= 13
H= 28

Superficie del Cilindro de radio 13 y altura 28 = 3348.95

R= 1
H= 0

Superficie del Cilindro de radio 1 y altura 0 = 6.2832

R= 0
Ok
```

CAPITULO 7: BUCLES Y NIDOS

Estudiamos en este capítulo las sentencias múltiples siguientes:

— FOR-NEXT
— WHILE-WEND

EL CICLO FOR-NEXT

Hemos querido dejar para el final algo que en los libros de Basic suele estudiarse al principio. Y lo hemos hecho con toda intención, porque, a pesar de la frecuencia con que se emplea, el bucle FOR-NEXT, y aun más el WHILE-WEND, son sentencias *de lujo.*

Esto del lujo exige un inciso aclaratorio.

Llamamos "de lujo" a cualquier sentencia cuyo cometido pueda ser llevado a cabo por el Basic mediante el empleo de sentencias "más primitivas", aunque pagando el tributo de un esfuerzo adicional.

(En nuestra manera cotidiana de expresarnos sucede otro tanto. "Bosque" puede sustituirse por "agrupación de árboles". Y "arbol" por "madera más o menos cilindrica que en su parte superior etc.". Y así sucesivamente. No cabe duda de que nuestro hablar sería más laborioso. Pero también es cierto que nuestro vocabulario podría reducirse enormemente).

En todo lo que hemos visto hasta ahora han surgido algunas sentencias de este tipo. Por ejemplo, el ELSE que en ocasiones acompaña al IF podría no ser utilizado nunca. Siempre hay modos de manejar instrucciones condicionales, por complicadas que sean, sin tener que incluir el ELSE.

Asimismo, LEFT$ también podría evitarse: para cualquier valor de A$ y de X, LEFT$(A$,X) siempre puede ser sustituida por MID$(A$,1,X) con idéntico resultado.

Y también SPACE$(N) es sustituible por STRING$(N," "). Y

aun esta última —"STRING$"— entra dentro de las de lujo. Podríamos trabajar sin que existiera (aunque con un mayor esfuerzo de programación).

Si quisiéramos que en la variable REP$ por ejemplo, entrara una cadena de 122 "X" iguales sólo tendriamos que poner:

```
250 REP$="":I=1
255 REP$=REP$+"X":I=I+1: IF I>122 THEN 270
260 GOTO 255
270 PRINT REP$
```

Si ejecutamos el programa anterior veriamos que REP$ es igual a una cadena de 122 "X". Evidentemente, es más corto y fácil poner:

```
250 REP$=STRING$(122,"X")
```

Incluso la importante sentencia GOSUB entraría dentro de las "de lujo"; sólo tendríamos que repetir las instrucciones que componen la rutina tantas veces como sea necesario. (Otra cosa será que la capacidad de la memoria interna permita semejantes dispendios).

El ciclo (o bucle o lazo) FOR-NEXT es un recurso muy cómodo cuando se tiene que repetir una o varias instrucciones, cosa que sucede con harta frecuencia.

Empezamos con el sencillo ejercicio de sacar por pantalla los N primeros números enteros. Conforme a lo hasta ahora estudiado, pondriamos:

```
10 INPUT N
20 CONTADOR=1
30 IF CONTADOR>N THEN  END
40 PRINT CONTADOR
50 CONTADOR=CONTADOR+1
60 GOTO 30
```

Con el ciclo FOR-NEXT bastaría con poner:

```
10 INPUT N
20 FOR I=1 TO N
30 PRINT I
40 NEXT I
```

O también —usando una sola línea—:

```
10 INPUT N:FOR I=1 TO N:PRINT I:NEXT I
```

Traduciendo al castellano diríamos: "Para I=1 (O "desde que I vale 1") hasta que valga N, pinta I". La sentencia NEXT (que significa «próximo», «siguiente») no sirve más que para cerrar el lazo. Naturalmente no tiene ningún sentido por sí sola. Necesita estar *precedida* de una sentencia FOR. Igualmente, una sentencia FOR no tiene tampoco sentido si no tiene un NEXT *posterior.*

Gran parte de los programas-ejercicio que hemos presentado hasta el momento son susceptibles de utilizar un ciclo FOR-NEXT.

Su estructura más general es:

```
FOR Var.Num. = Primer Valor TO Limite Sup. [STEP Paso]
```

En la que:

— lo que figura en negrita debe aparecer exactamente así.

— "Var. Num." representa una variable numérica entera o de simple precisión (el compilador admite que pueda ser también de doble precisión) que irá cambiando de valor a medida que se ejecuta el ciclo.

— "Primer Valor", es el valor que recibe la "Var. Num." al comenzar el ciclo. Puede ser una constante, una variable o una expresión numérica.

— "Límite Sup.", es otra constante, variable o expresión numérica que indica el limite superior que puede recibir la «Var.Nú.» *sobrepasado* el cual, el programa continua su ejecución en la instrucción *siguiente* al NEXT.

— "Paso" es otro valor numérico que establece el incremento que recibe el valor de la "Var. Num." en cada vuelta del ciclo. Viene precedido de la palabra STEP (paso, escalón) y lo hemos puesto entre corchetes queriendo indicar con ello que puede omitirse. Cuando no se pone, el "paso" por defecto es la unidad.

Vamos a poner una serie de ejemplos en los que aparezcan todas la peculiaridades posibles de estas sentencias. El simple análisis del resultado de estos programitas nos excusarán en buena medida de largas explicaciones.

```
5 INPUT A$:IF A$="F" THEN END
10 FOR A=1 TO LEN(A$)
20 PRINT LEFT$(A$,A)
30 NEXT A

RUN <CR>
?ALICIA
A
AL
ALI
ALIC
ALICI
ALICIA
?F
Ok

5 INPUT A$:IF A$="Q" THEN END
10 FOR N=1 TO LEN(A$) STEP 5
20 PRINT LEFT$(A$,N):NEXT N
30 PRINT:GOTO 5

RUN <CR>
?ALEJANDRO CABALLERO GUERRA
A
ALEJAN
ALEJANDRO C
ALEJANDRO CABALL
ALEJANDRO CABALLERO G
ALEJANDRO CABALLERO GUERRA

?ABCDEFGHIJKLMNOPQRSTUVWXYZ123
A
ABCDEF
ABCDEFGHIJK
ABCDEFGHIJKLMNOP
ABCDEFGHIJKLMNOPQRSTU
ABCDEFGHIJKLMNOPQRSTUVWXYZ
?0
Ok
```

En este ejemplo observamos que en la ejecución del FOR-NEXT sobre el último INPUT, alguien nos ha escamoteado el "123" del final. No podía ser de otra manera. Porque la cadena introducida tiene una "LEN" (una longitud) de 29 caracteres. Entonces, la instrucción:

```
FOR  N=1 TO LEN(A$) STEP 5 es lo mismo que FOR N=1 TO 29 STEP  5
```

Con ello, N, al comenzar el ciclo, toma el valor 1 y, tras ejecutar la instrucción de PRINT, alcanza el NEXT con lo que N se incrementa en 5 (es decir, adquiere el valor 1+5=6), ... luego vale 11, 16, 21, 26, ... 31?. No, No llega a 31. El último valor que recoge es 26 porque el tope (LEN(A$)) está en 29. Al ir a incrementarse ese 26 y encontrarse con un 31 superior al tope establecido, el bucle se interrumpe, saltando la secuencia del programa a la sentencia que sigue al "NEXT".

Vamos con otros ejemplos.

```
20 FOR I=100 TO 74 STEP -10
40 PRINT I;:NEXT:PRINT "NO HAY MAS"

RUN <CR>
 200  180  160  NO HAY MAS
Ok

10 FOR I=10 TO 5:PRINT "NO VA A SALIR NADA":NEXT

RUN <CR>
Ok

10 A=20:B=5
20 FOR R=2*A-3*B TO 2*B-3*A+20 STEP -B
30 PRINT R;
40 NEXT R

RUN <CR>
 25  20  15  10  5  0 -5 -10 -15 -20 -25 -30
Ok

10 FOR A=1 TO 26
20 PRINT CHR$(A+64);
30 NEXT
```

```
RUN <CR>
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok

10 FOR DATA=48 TO 57 STEP 2:PRINT CHR$(DATA):NEXT

RUN <CR>
Suntax Error in 10
```

En efecto, el Basic tiene perfecto derecho a protestar. Porque DATA es una palabra reservada (es una sentencia) y no se puede utilizar como variable.

Observe el lector que en los dos últimos ejemplos hemos puesto el "NEXT" a pelo, sin adjuntarle la variable del FOR. Esto es admisible. En la práctica, cuando la programación se complica, esta omisión puede hacer perder claridad al programa al desaparecer la pista de las variables.

```
10 FOR I=1 TO 24
20 PRINT SPACE$(80)
30 NEXT I

RUN <CR>
```

(La pantalla queda limpia. Sólo un tímido Ok abajo).

Esta es una forma —no la mejor, desde luego— de borrar la pantalla. En le próximo capítulo estudiaremos mejores procedimientos.

```
10 FOR I=1 TO 1000:NEXT I

RUN <CR>
Ok
```

(¿Y esto qué es?. Aparentemente, nada. En realidad es un bucle vacío que lo único que hace es perder tiempo. Pero hay ocasiones en que es preciso perderlo. Lo veremos).

```
10 DATA 23,1,-4,12,3,5,1,-7,11,12,87,2,1,0,0,22,6,3,0,20
20 FOR N=1 TO 20
30 READ A:IF A>10 THEN D=D+1 ELSE IF A<0 THEN NN=NN+1
40 NEXT N
50 PRINT "HAY";D;"NUMEROS MAYORES DE 10"
60 PRINT "HAY";NN;"NUMEROS NEGATIVOS"
```

```
RUN <CR>
HAY 7 NUMEROS MAYORES DE 10
HAY 2 NUMEROS NEGATIVOS
Ok


10 FOR I=1 TO 1000
20 DA=INT(RND$6)+1:A(DA)=A(DA)+1
30 NEXT
40 PRINT:PRINT I
50 FOR I=1 TO 6
60 PRINT "EL";I;"HA SALIDO";A(I);"VECES"
70 NEXT I
80 PRINT:PRINT I

RUN <CR>
 1001
EL 1 HA SALIDO 184 VECES
EL 2 HA SALIDO 202 VECES
EL 3 HA SALIDO 150 VECES
EL 4 HA SALIDO 148 VECES
EL 5 HA SALIDO 160 VECES
EL 6 HA SALIDO 156 VECES

 7
Ok
```

Obsérvese los valores de la variable I, (la misma variable en los ciclos), a la salidad de éstos (1001 y 7). Una unidad por encima de los límites de los bucles. Totalmente lógico. La variable va incrementando en cada vuelta una unidad (que es el valor del STEP por defecto en nuestro caso) y sólo cuando ese valor sobrepasa el límite, se escapa del ciclo considerando que ha terminado su función. Pero la variable ha quedado cargada con el valor en exceso sin el cual no habría podido escapar del ciclo.

```
10 FOR I=1 TO 1000 STEP 1000
20 PRINT "PP"
30 NEXT I
40 PRINT I

RUN <CR>
```

```
PP
 2000
Ok


10 FOR I=1000 TO 1
20 PRINT "PP"
30 NEXT
40 PRINT I

RUN <CR>
 1000
Ok


10 FOR H=1 TO 2 STEP -1
20 PRINT "HOLA, MAJETE";
30 NEXT
```

He aquí un caso singular. No va a salir nada. Enseguida volvemos sobre esto.

Otro caso singular es este:

```
10 FOR I=1 TO 2 STEP 0
20 PRINT I;
30 NEXT
```

Nos cansaríamos de ver salir por la pantalla una serie interminable de "1" teniendo que dar CONTROL-C para abortar.

```
10 FOR I=1 TO 10000
20 A=RND*10000
30 IF A<1 THEN 100
40 NEXT I
100 PRINT I,A

RUN <CR>
 3862         .735866
Ok
```

Fíjese el lector con qué elegancia se ha interrumpido el ciclo cuando este iba por la vuelta 3862, obedeciendo a la condición que figura en la línea 30. Esto es de gran importancia. El cumplimiento de una condición *dentro* del propio bucle puede provocar la interrupción y salida del mismo sin contemplaciones. (Esto no ocurre en todos los Basic).

El mismo ejercicio anterior, pero repitiendo 5 veces la misma historia:

```
5  N=N+1:IF N>5 THEN END
10 FOR I=1 TO 10000
20 A=RND*10000
30 IF A<1 THEN 100
40 NEXT I
100 PRINT I,A
110 GOTO 5

RUN <CR>
 3362          .735866
 2169          .621432
 10001         3956.5
 388           .0683229
 10001         3152.96
Ok
```

¿Y no apetecería utilizar otro bucle que envolviese al que se encuentra horquillado entre las líneas 20 y 40 y ahorrarnos el incómodo contador que hay en la 5?

Acabamos de inventar el "FOR-NEXT *anidado*". El programa anterior se convertiría en:

```
5  FOR N=1 TO 5
10   FOR I=1 TO 10000
20        A=RND*10000
30        IF A<1 THEN 100
40   NEXT I
100  PRINT I,A
110 NEXT N
```

Con idéntico resultado en su ejecución. Señalamos al lector la posición de los "NEXT" respecto a los "FOR".

Pero antes de meternos en mayores complicaciones y a la vista de los resultados de los anteriores programas-ejemplo, vamos a analizar el funcionamiento del ciclo FOR-NEXT simple.

La variable del FOR toma *siempre* el primer valor. A continuación se produce un contrastado de ese valor con el límite marcado por el "'TO" y con el valor del STEP. Funciona entonces la lógica más estricta: si el limite del "TO" es inferior al primer valor y el del STEP es positivo —o el límite es superior al primer valor y el del STEP es negativo— entonces el programa se sale automáticamente del ciclo, saltando a la primera instrucción que encuentre *después* del NEXT. Si no encuentra ninguna, porque el NEXT se halle al final del programa, éste se autoacaba, como es su costumbre.

Cada vez que el programa alcanza un NEXT, se produce un incremento de la variable del ciclo del valor del STEP, que, por defecto, es la unidad.

Las instrucciones que se encuentran entre el FOR y el NEXT se ejecutan de acuerdo con la propia lógica que tengan.

PR- 1.39: Queremos ver salir por pantalla todos los números impares del 1 al 200 que *no sean* múltiplos de 3 ni de 7.

```
10 FOR A=1 TO 200 STEP 2
20 IF A/3=INT(A/3) OR A/7=INT(A/7) THEN 40
30 PRINT A;
40 NEXT A

RUN <CR>
 1  3  5  11  13  17  19  23  25  29  31  37  ...
Ok
```

PR- 1.40: Se trata de un test de cálculo aritmético, suma de dos números, cuya magnitud es función de la edad del operador (el sujeto del test).

```
40 INPUT "DIME TU EDAD (0 =FIN):",EDAD
60 IF EDAD=0 THEN END
70 ACIERTOS=0
80 FOR I=1 TO 10
100 A=INT(RND*EDAD*3):B=INT(RND*EDAD*3)
120 PRINT A;"+";B;"=";
140 INPUT S
160 IF S=A+B THEN ACIERTOS=ACIERTOS+1
180 NEXT I
200 PRINT:PRINT ACIERTOS;" ACIERTOS":GOTO 40
```

PR- 1.41: Programa de ortografía. El operador va a efectuar un test en el que deberá escribir —sin CR— una "b" o una "v" que correspondan a las palabras que vayan saliendo en las que en lugar de la "b" o la "v" hay un asterisco. Tendra un limite de tiempo para hacerlo según su edad.

```
10 INPUT "EDAD QUE TIENES (0=FIN):",EDAD
20 IF EDAD=0 THEN END
22 IF EDAD>70 THEN PRINT"SOPAS Y BUEN VINO":GOTO 10
30 DATA *ECERRO,*ARCO,*UELO,A*UELO,A*OLIR,HIER*E,
   YER*A,HA*A,LA*IO,NU*E,SA*OTAJE,HIL*ANAR,AL*EDLO,
   LIE*RE,U*RE,SA*IO,LI*RO,*IENTO,*ENTA,*ENEVOLO
40 CLAVE$="BBVBBVBBBBBVVBBBBVVB":RESTORE 30
45 ACIERT=0
47 RESTORE
50 FOR P=1 TO 20
55 READ A$:PRINT A$
60 FOR LAPSO=1 TO 4*(90-EDAD)
70 Y$=INKEY$:IF Y$="B" OR Y$="V" OR Y$="b"  OR  Y$="v"
   THEN 90
80 NEXT LAPSO:GOTO 100
90 IF  (Y$="B" OR Y$="b") AND MID$(CLAVE$,P,1)="B"  OR
   (Y$="V" OR Y$="v") AND MID$(CLAVE$,P,1)="V"
   THEN ACIERT=ACIERT+1
100 NEXT P
110 PRINT ACIERT;" ACIERTOS"
120 IF ACIERT<20 THEN PRINT "INTENTALO OTRA  VEZ":GOTO 45
130 PRINT "*IEN, MUY *IEN":PRINT:GOTO 10
```

EL CICLO WHILE-WEND

De uso mucho más restringido que el FOR-NEXT, tiene aplicaciones específicas bastante fecundas.

Hay que empezar por dejar sentado que cualquier bucle WHILE puede ser sustituido fácilmente por otro FOR. Si deciamos que éste era de lujo, el WHILE lo es con más razón.

En castellano, el "WHILE (la condición que sea)" se traduce por "MIENTRAS (la condición que sea)". Detrás del WHILE (con su condición), y antes del WEND (final de ciclo), vendrá una instrucción o una serie de ellas. Pues bien, "mientras" se cumpla la

condición que acompaña al WHILE, todas las instrucciones que pueda haber hasta el WEND se ejecutan a pies juntillas. Y en el momento en que deje de verificarse dicha condición, entonces, inexorablemente el programa continuará en la instrucción *siguiente* al WEND ("Fin del WHILE").

Resaltemos las diferencias con el ciclo FOR-NEXT. Con éste, las instrucciones comprendidas entre el FOR y el NEXT, se ejecutaban un número determinado de veces: las indicadas por la propia sentencia FOR ("FOR I=1 TO 100", cien veces). Ya sabemos que todo es relativo, que se puede interrumpir este ciclo, si una instrucción interior ("IF I>12 GOTO 680") así lo reclama.

Con el WHILE-WEND, en cambio, no se especifica el número de "vueltas" que va a dar el bucle. No es un número absoluto (como lo era en el FOR-NEXT) sino *relativo*, relacionado con la condición exigida por el WHILE.

Como dirían los chinos, más vale un programa que mil explicaciones.

Pero como hemos dicho que ambos ciclos son intercambiables, vamos a resolver un mismo problema con una y otra herramienta.

PR- 1.42: Queremos conseguir (y ver) los 10 primeros números aleatorios mayores de 99 centésimas, que el ordenador es capaz de generar utilizando el lazo FOR-NEXT.

```
10 FOR I=1 TO 30000
20 A=RND:IF A>.99 THEN N=N+1
30 IF N>100 THEN 70
40 PRINT A
50 NEXT
70 PRINT
80 PRINT "HEMOS TERMINADO"
```

Vea el lector que en la línea 10 hemos tenido que poner un número muy alto para tener la seguridad de que conseguiremos encontrar 10 números que cumplan con ello.

PR- 1.43: El mismo programa anterior pero empleando el ciclo WHILE-WEND.

```
5 A=0
10 WHILE A<=.99
20 A=RND:WEND
30 PRINT A:N=N+1
```

```
40 IF N<10 THEN 5
50 PRINT
60 PRINT "HEMOS TERMINADO"
```

Y todavía, en plan sofisticado, podemos probar:

PR- 1.44: Y el mismo programa sólo con WHILEs.

```
10 WHILE N<10
20 A=0
30   WHILE A<=.99
40   A=RND
50   WEND
60 PRINT A:
65 N=N+1
70 WEND
```

(Como se ve, el WHILE también se puede "anidar" con toda soltura).

Traducimos a un pseudocastellano el programa anterior:

"Mientras N sea menor de 10 (al inicio del programa vale cero), tú, variable A, hazte cero. Y en estas condiciones, mientras A sea menor de 99 centésimas, que en A se cargue un número aleatorio entre 0 y 1, todas las veces que haga falta. Cuando A no cumpla con el segundo «mientras», que lo saque en pantalla y que el contador N aumente en una unidad. Y que cuando este contador N no cumpla con el primer "mientras", termine".

Espantosa traducción pero nítida lógica. El empleo de bucles WHILE produce una especial limpieza en la programación. En concreto, el miniprograma anterior se podría escribir en una sola línea:

```
10 WHILE N<10:A=0:WHILE A<=.99:A=RND:WEND:PRINT A:N=N+1:WEND
```

Al lector que esté dispuesto a ser muy exigente consigo mismo, le recomendamos que intente repetir todos los programas que han surgido en este y anteriores capítulos utilizando (en los que se pueda) el ciclo FOR-NEXT. Y si su exigencia le da para ello, que haga otro tanto con el ciclo WHILE-WEND.

CAPITULO 8.— LA PANTALLA

En todo ordenador, del más grande al más pequeño, hay dos periféricos que no pueden faltar:

— el Teclado, periférico de *entrada* por excelencia y
— la Pantalla, periférico de *salida* por idem.

Mediante el teclado, "hablamos" al ordenador.
Mediante la pantalla, el ordenador "nos habla".

Y la pantalla nos habla tanto, que incluso nos hace aparecer en ella lo que vamos introduciendo por el teclado. Está claro, pues, que este periférico requiere una atención muy especial.

En este capítulo vamos a abordar dos cuestiones relacioneadas con el dominio de la pantalla: el *dónde* van a salir las cosas (la pantalla es muy grande, en una estándar caben unos dos mil caracteres) y el *cómo* deben aparecer (hay que conseguir que la informática no esté reñida con las normas estéticas elementales).

La pantalla, como la de un televisor, está organizada en puntos ("pixels"). Cada carácter visualizado en pantalla, cualquiera que sea su tamaño, ocupa una porción fija de pixels en horizontal y vertical. Es como si la pantalla formase una cuadrícula invisible en la que van a encajar todos los caracteres que aparezcan en ella. En una pantalla normal caben 24 líneas de 80 caracteres (1920 en total).

Resulta entonces que cada carácter, cada signo, cada letra sólo puede aparecer en una de esas 1920 posiciones prefijadas. Cada posición queda perfectamente determinada por dos números (dos coordenadas):

— Posición horizontal contada desde el borde izquierdo de la pantalla cuyo valor irá del 1 al 80 (o del 0 al 79).
— Posición vertical contada desde el borde superior de la pantalla: del 1 al 24, o del 0 al 23 en nuestra pantalla estándar.

El establecimiento de estas dos coordenadas va a permitirnos colocar cualquier cosa en una deseada situación.

Hasta el momento sólo las sentencias PRINT e INPUT nos permiten situar cosas en la pantalla. Menguados pinceles para nuestras almas de artista.

Cada vez que empleamos un PRINT en un programa se produce un salto de línea a la inmediata inferior. Si no queríamos que este salto se produjese, haciamos "sujetar" al cursor adosando al final de la instrucción de PRINT, un ; (punto y coma) o una , (coma) con el resultado que ya conocemos.

Con la sentencia INPUT, y en lo que se refiere a la pantalla, las cosas eran parecidas.

En cualquier caso, el salto efectuado como consecuencia del empleo de cualquiera de las dos sentencias anteriores, sitúa al cursor *al principio* de la línea siguiente, es decir, completamente a la izquierda. Cuando, con estos saltos descendentes, llegamos a la parte más baja de la pantalla, se produce el "scrolling" que permite seguir visualizando línea tras línea (al tiempo que van desapareciendo las líneas de arriba).

TAB

Con la función TAB podemos desplazarnos horizontalmente en la Pantalla (y también en la impresora).

Si queremos que aparezca la palabra "Sonata" separada diez posiciones del borde izquierdo de la pantalla, podemos poner:

```
PRINT "          ";"Sonata"
```

O también:

```
PRINT "            Sonata"
```

O incluso:

```
PRINT SPACE$(10);"Sonata"
```

Mejor:

```
PRINT TAB(10);Sonata"
```

TAB es abreviatura de "TABULACION". Tabular es sinónimo de "situar horizontalmente".

La instrucción:

```
300 PRINT TAB(12);"FATIMA";TAB(30);"ESPERANZA";TAB(50);"ALICIA"
```

produciría con su ejecución la salida en pantalla de:

```
        FATIMA          ESPERANZA          ALICIA
```

Para que esos mismo tres nombres apareciesen en forma análoga en impresora, bastaría con cambiar la sentencia PRINT por la LPRINT.

Intente el lector adelantarse al resultado de la ejecución del siguiente programa (si le da mucha pereza, le aconsejamos al menos que se concentre en lo que ocurre en las instrucciones 200-230 de rica aplicación en BASIC).

```
10 READ A$
20 IF A$="YA VALE" THEN 100
30 PRINT TAB(15);A$
40 GOTO 10
50 '
100 RESTORE
110 READ A$:IF A$="YA VALE" THEN 200
120 PRINT TAB(LEN(A$));A$
130 GOTO 110
140 '
200 RESTORE
210 READ A$:IF A$="YA VALE" THEN 300
220 PRINT TAB(40-LEN(A$));A$
230 GOTO 210
240 '
300 RESTORE:B$=" "
310 READ A$:IF A$="YA VALE" THEN END
330 PRINT TAB(LEN(B$));A$
335 B$=B$+A$
340 GOTO 310
350 '
1000 DATA ERNESTO,EMI,HORTENSIA,ALEJANDRO,INES,YA VALE


RUN <CR>
                ERNESTO
                EMI
                HORTENSIA
                ALEJANDRO
```

```
                INES
        ERNESTO
    EMI
                HORTENSIA
                ALEJANDRO
     INES
                                    ERNESTO
                                       EMI
                                   HORTENSIA
                                   ALEJANDRO
                                       INES
ERNESTO
        EMI
           HORTENSIA
                   ALEJANDRO
                           INES
Ok
```

Si hacemos el programa del capítulo 6 que nos saque en pantalla los cuadrados y los cubos de los N primeros números naturales (siendo N un número dado por INPUT dentro del programa), podríamos enforcarlo así:

```
10 INPUT "ESCRIBE UN NUMERO ENTERO: ",N
20 PRINT
25 IF N=0 THEN END
26 '
30 PRINT "Nro.";TAB(10);"Cuadrados";TAB(30);"   Cubos    "
40 PRINT "----";TAB(10);"---------";TAB(30);"-----------"
50 FOR I=1 TO N
60 PRINT I;TAB(10);I*I;TAB(30);I*I*I
70 NEXT I
80 PRINT:GOTO 10
```

La ejecución de este programa produciría esta salida (bastante fea, dicho sea sin ánimo de ofender).

```
RUN <CR>
ESCRIBE UN NUMERO ENTERO: 10
```

```
Nro.        Cuadrados          Cubos
----        ---------          -----------
1           1                  1
2           4                  8
3           9                  27
4           16                 64
5           25                 125
6           36                 216
7           49                 273
8           64                 512
9           81                 729
10          100                1000

ESCRIBE UN NUMERO ENTERO: 0
Ok
```

El núcleo del programa está en la línea 60:

```
60 PRINT I;TAB(10);I*I;TAB(30);I*I*I
```

Cuando se trata de tabular algo numérico, la justificación debe hacerse *por la derecha.* Esto lo podemos conseguir (no de la mejor forma, desde luego) transformando I,I*I e I*I*I en algo que cumpla con esa condición. Basta para ello cambiar la instrucción 60 por:

```
60 PRINT TAB(4-LEN(STR$(I)));I;
62 PRINT TAB(17-LEN(STR$(I*I)));I*I;
64 PRINT TAB(37-LEN(STR$(I*I*I)));I*I*I
```

Con el resto del programa intacto, su ejecución daría:

```
RUN <CR>
ESCRIBE UN NUMERO ENTERO: 10

Nro.        Cuadrados          Cubos
----        ---------          -----------
 1           1                  1
 2           4                  8
 3           9                  27
 4           16                 64
 5           25                 125
```

```
 6          36          216
 7          49          273
 8          64          512
 9          81          729
10          100         1000

ESCRIBE UN NUMERO ENTERO: 0
Ok
```

PRINT USING

Aun se puede mejorar la disposición en pantalla o impresora de los datos numéricos utilizando la sentencia PRINT USING ("pinta utilizando"). La mala traducción sugiere la necesidad de algo mas (se entiende que "utilizando" necesita "algo" a su lado). Ese algo es una *máscara*, una especie de horma o molde donde los valores numéricos van a quedar perfectamente encajados, de acuerdo con aquél.

El molde-máscara se diseña con signos # , comas y puntos, habida cuenta de que unas y otros tienen una aplicación inversa a lo que aprendimos de zagales: la "coma decimal" pasa a ser el "punto decimal" y el punto de millares se convierte en la coma de millares. Si a nuestro lector no le agrada desprenderse de sus costumbres, lea este epígrafe con una miaja de displicente escepticismo, y siga esforzándose en busca de más fértiles vegas.

La máscara tras el USING puede adoptar multitud de formas. Una es, por ejemplo, esta: " # # # # # ".

```
20 DATA 12345,12,123,123456,1234,1
40 FOR I=1 TO 6
50 READ A
70 PRINT USING "######";A
80 NEXT

RUN <CR>
 12345
    12
   123
123456
  1234
     1
Ok
```

El mismo programa, con la instrucción 70 de otra guisa:

```
70 PRINT USING "###,###.##"


RUN <CR>
 12,345.00
     12.00
    123.00
123,456.00
  1,234.00
      1.00
Ok
```

La máscara puede ser cargada previamente en una variable alfanumérica:

```
10 MASC$="###,###.##"
- - -....
70 PRINT USING MASC$;A
- - -
```

Otras muchas variantes tiene esta sentencia, que ya escapan a los límites de esta Primera parte.

USO DE LAS COORDENADAS COMPLETAS (HORIZONTAL Y VERTICAL)

Si lo que deseamos situar en pantalla es algo relacionado no con una línea , sino con unas coordenadas concretas, entonces es otro cantar.

Muchos BASIC disponen de una instrucción particular que obliga al cursor a posicionarse en la horizontal y vertical que acompaña a esta particular sentencia. Damos algunos ejemplos de esos BASIC:

```
APPLESOFT:     HTAB 5:VTAB 7:PRINT "HELIO"

BASIC IBM:     LOCATE 7,5:PRINT "HELIO"

HP-250   :     CURSOR (5,7):PRINT "HELIO"
```

En todos los casos obtenemos el mismo resultado. El BASIC de Microsoft resuelve el problema de un modo, a primera vista más complejo y sofisticado. Pero hay que introducirse en una programación seria para darse cuenta de que su procedimiento de situar el cursor (y por tanto el mensaje que se desee), es en la práctica mucho más eficaz. Menos infantil, diríamos. Para abordar este tema con la profundidad que merece, tenemos que hacer un inciso, con algunas ramificaciones, bastante útiles, en verdad.

Secuencias de Escape

Cuando estudiabamos los caracteres ASCII, deciamos que el que corresponde al 27 (precisamente al 27), o sea, el *CHR$(27),* es un carácter de ESCAPE que funciona a modo de "modificador" del carácter (o los caracteres) que le siguen. Y esos caracteres modificados por el de Escape son específicos para cada ordenador. No corresponde a nuestro BASIC el decir cuáles son. Son caracteres "machine dependent" (dependientes de la marca del ordenador con el que nos hallemos trabajando). Esas cadenas (casi siempre cortas) componen las llamadas "Secuencias de Escape". Estas son las más características:

- Posicionamiento del cursor.
- Borre de pantalla.
- Insertado de Línea.
- Borrado de Línea.
- Comienzo de alta intensidad.
- Comienzo de baja intensidad.
- Comienzo de video inverso.
- Comienzo de video subrayado.
- Borre de lo que esté en alta intensidad.
- Comienzo de escritura con letra expandida en impresora.
- Comienzo de escritura con letra comprimida en idem.
- Comienzo de escritura con letra normal en idem.

(Estas tres últimas secuencias de escape, evidentemente, no vienen exigidas por el ordenador sino por la propia impresora a cuyas órdenes va a trabajar ésta como periférico de aquel. Diremos también, de paso, que hay impresoras en las que la "orden" de cambio de tamaño de letra no se produce tras una verdadera secuencia de escape sino tras un simple carácter. Caso concreto: el de algunos modelos de SHARP en los que la simple instrucción:

```
LPRINT CHR$(15)
```

hace salir todo lo que a continuación se imprima en caracteres comprimidos).

La estructura general de todas las secuencias de escape es:

```
PRINT CHR$(27)+SEC$
```

en la que SEC$ expresa el o los caracteres que cada fabricante haya diseñado como acompañante del carácter de escape para cada uno de los objetivos antes reseñados.

La forma práctica de actuar es:

1ro.: consultar los manuales de operación del ordenador donde aparecerán los SEC$ correspondientes a cada objetivo de pantalla.

2do.: disponer una cabecera de programa (con ella darán comienzo todos nuestros programas de aplicación) en donde cargaremos en variables nemotécnicas cada una de las secuencias de escape que vayamos a utilizar. Por ejemplo:

```
10 E$=CHR$(27):BP$=E$+"R":AI$=E$+"G0":BI$=E$+"G4" Etc etc.
```

en la que el programador ha elegido BP$ como "Borre de Pantalla", AI$ como "Alta Intensidad", etc. A lo largo de los programas encabezados por esa línea 10, cada vez que aparezca la instrucción PRINT BP$, se producirá un borrado completo de la pantalla. Y así las demás. Antes de que nuestro paciente lector empiece a abominar del BASIC de Microsoft ante tamaños horrores, queremos asegurarle que, en la práctica utilizará con verdadero deleite las variables nemotécnicas que él mismo se habrá preparado. De hecho acabará olvidando el contenido real de todas esas variables sin que nadie sufra por ello.

Digeridos los anteriores comentarios, podemos concentrarnos ya en el posicionamiento del cursor.

En la asignación que sigue, CURSOR$ quedaría cargado con cuatro caracteres cuyo contenido daría la *posición del cursor.*

```
CURSOR$=CHR$(27)+PC$+CHR$(V+32)+CHR$(H+32)
```

en la que:

PC$ es un carácter "machine dependent", específico del ordenador con el que estemos trabajando, y que habrá de ser definido previamente.

V es el número de línea, cuyo valor estará comprendido entre 1 (cabecera de pantalla) y 24 (pie de pantalla).

H es el número de columna, cuyo valor será 1 en el borde izquierdo de la pantalla y 80 en el derecho.

Si ejecutamos el siguiente miniprograma:

```
100 H=10:V=10:PC$="Y"
105 ' La "Y" es el caracter que corresponde a la màquina
106 ' particular con la que trabajamos.
110 CURSOR$=CHR$(27)+PC$+CHR$(V+32)+CHR$(H+32)
120 PRINT CURSOR$;"Luis Candelas"
```

Veremos aparecer al generoso bandido en las coordenadas 10,10.

Es evidente que si no buscamos solución mejor, la programación se va a hacer poco menos que insoportable, si cada vez que queramos situar algo en pantalla tenemos que largar toda esa ristra de aparentes sinsentidos.

La mejora podría consistir en utilizar la secuencia de caracteres "posición del cursor" como rutina general:

```
100 H=10:V=10
110 GOSUB 1000
120 PRINT "Luis Candelas"
- - -
- - -
1000 PRINT CHR$(27);"Y";CHR$(V+32);CHR$(H+32);:RETURN
```

Esto ya es otra cosa. Pero sigue siendo un poco latoso.

He aquí la solución definitiva (¡benditas funciones definibles!):

```
10 DEF FNC$(H,V)=CHR$(27)+"Y"+CHR$(V+32)+CHR$(H+32)
20 '
- - -
- - -
100 PRINT FNC$(10,10);"Luis Candelas"
```

Con el truco de haber definido al comienzo del programa la función de posicionamiento del cursor (lo cual nos permite no tener ni que recordar cómo nació el precioso FNC$), logramos no sólo ya disponer de una buena y manejable herramienta; es que pode-

mos incluso burlarnos de los orgullosos "LOCATE" de IBM y las otras sentencias que antes apuntábamos. Porque con nuestro arduo hallazgo del FNC$, disponemos de algo realmente valioso.

Veámoslo con un ejemplo tan frecuente como diáfano.

Queremos que salga en pantalla la ficha de nuestro Documento Nacional de Identidad, precisamente así:

```
Numero:          Apellidos:
                 Nombre:

Nacido en:                       Provincia:
Fecha Nac.:

N. Padre:                        N. Madre:

Domicilio:
Poblacion:
Provincia:

Expedido en:                     Prov.:
Fecha Expedicion:
```

Con la definición de FNC$ del fragmento de programa anterior, éste sería el trozo de programa que produciría la salida en pantalla en las posiciones deseadas:

```
290 WIDTH 255
300 PRINT FNC$(11,5);"Numero:";FNC$(28,5);"Apellidos:"
310 PRINT FNC$(28,6);"Nombre:";FNC$(11,8);"Nacido en:"
320 PRINT FNC$(41,8);"Provincia:";FNC$(11,9);"Fecha Nacto.:"
330 PRINT FNC$(11,11);"N. Padre:";FNC$(41,11);"N. Madre:"
340 PRINT FNC$(11,13);"Domicilio:";FNC$(11,14);"Poblacion:"
350 PRINT FNC$(11,15);"Provincia:";FNC$(11,17);"Expedido en:"
360 PRINT FNC$(41,17);"Prov.:";FNC$(11,18);"Fecha Expedicion:"
```

Con el sistema IBM (o cualquier análogo en el que existiera una sentencia específica de posicionamiento del cursor) habría que poner (nos limitamos a la salida equivalente a la línea 300 anterior):

```
300 LOCATE 5,11:PRINT "Numero:":LOCATE 5,28:PRINT "Apellidos:"
```

La cosa parece estar en empate. Pero la discusión queda dirimida a favor de nuestro FNC$ en cuanto hablemos de disponer de una variable alfanumérica que tenga cargada la pantalla completa, como si de un mensaje compacto se tratase:

```
290 WIDTH 255
300 PA$=FNC$(11,5)+"Numero:"+FNC$(28,5)+"Apellidos:"
310 PA$=PA$+FNC$(28,6)+"Nombre:"+FNC$(11,8)+"Nacido en:"
320 PA$=PA$+FNC$(41,8)+"Provincia:"+FNC$(11,9)+"Fecha Nac.:"
330 PA$=PA$+FNC$(11,11)+"N. Padre:"+FNC$(41,11)+"N. Madre:"
340 PA$=PA$+FNC$(11,13)+"Domicilio:"+FNC$(11,14)+"Poblacion:"
350 PA$=PA$+FNC$(11,15)+"Provincia:"+FNC$(11,17)+"Expedido en:"
360 PA$=PA$+FNC$(41,17)+"Prov.:"+FNC$(11,18)+"Fecha Expedicion:"
```

De esta astuta manera, disponemos de una variable, PA$, donde se encuentra cargada la ficha completa (formato pantalla tal como nos haya parecido mejor situar los campos). Si no la tocamos a lo largo del programa, ahí la tiene Vd. para lo que guste. Y si prevé que va a trabajar con varias fichas, tampoco resulta difícil organizar el vector PA$(*) para cargar en cada uno de sus elementos cada uno de las fichas (o "pantallazos" en general) con los que vayamos a trabajar. Todo ello con la única molestia de tener que cargar al principio de los programas con la anterior línea 10 en la que queda definida la función de posicionamiento del cursor.

CRITICA DE LA SENTENCIA INPUT

No podemos terminar este capítulo dedicado a la pantalla sin someter a una dura crítica a la sentencia INPUT que sólo debe emplearse en programas de pruebas y simulaciones. Nunca en programas serios. Es preciso buscar mejores modos de entrar en conversación con el programa. Y ello por las razones siguientes:

1.— Un INPUT en el que la variable que le acompaña sea numérica, no permite la introducción de datos alfanuméricos. No tenemos nada contra esta exigencia. Pero lo que no nos agrada en modo alguno es la *forma* con que el Basic rechaza estas entradas espúreas. Un error de este tipo hace saltar un abominable mensaje anglomorfo:

"Redo from start?"

(="Rehágalo desde el comienzo" o "Vuelva a introducir algo que sea correcto"). Pero lo malo es que, además, este horrendo mensaje surge en la línea siguiente (la de abajo) a aquella en la que estamos. Nos rompe por completo la estética de nuestro trabajo. En el ejemplo de la ficha del D.N.I. que antes veíamos, queda ésta completamente desbaratada. Vamos, un verdadero desastre.

2.— Cuando estamos introduciendo un dato, todos los blancos (espacios) que hayamos pulsado, al principio y/o al final del INPUT, nos son escamoteados con impertérrita frialdad en el momento de la introducción. Es decir, si introducimos

```
"  Robert Redford  "
```

en la variable alfanumérica que acompañe al INPUT quedará cargado

```
"Robert Redford"
```

Esto, algunas veces, produce o puede producir problemas. Porque hay ocasiones en que *necesitamos* que esos espacios iniciales o finales queden incorporados a la variable.

3.— Si en la variable del INPUT pretendemos que figure una coma o un punto y coma, nos quedaremos con las ganas. Esos signos de puntuación los interpreta el INPUT como *separadores*. Ejemplo clásico es la introducción de una dirección (las señas, el domicilio) en una ficha o registro:

```
Paseo de la Bona Nova, 22
```

Inútil. En la variable DIR$, donde pretendiamos que quedara incorporada esa dirección completa, sólo entrará "Paseo de la Bona Nova", sin la coma y sin el 22. Y encima, como el BASIC buscará donde cargar el "22" sin encontrar variable que llevarse a la boca, nos dirá impávidamente que hemos incurrido en error quedándose tan fresco.

(Hemos de decir, en honor a la verdad, que esta limitación del INPUT, queda subsanada mediante el empleo de la sentencia LINE INPUT que permite la introducción de separadores como otros caracteres cualesquiera).

4.— En todos los casos anteriores, el intérprete nos permite con imperceptible sarcasmo que nos "equivoquemos" (desde su particular punto de vista, a todas luces ilegítimo). No nos los dirá hasta que pulsemos CR. Cuando el dato introducido es largo, este comportamiento enciende la ira del más templado.

5.— En la paulatina introducción del dato, carácter a carácter, no se presenta ninguna limitación por parte del intérprete (p. ej., si el dato debe ser una fecha, podremos escribir "31-2-88" o incluso "3a-ju-&&". Y a lo mejor, si el programa no es correcto, puede que estos desmanes entren impunemente en el programa.

6.— Si damos CONTROL-C, tanto en intérprete como en compilado, el programa se nos aborta. Un programa depurado debe ser tan consistente como un bloque de granito. No debe ser "abortable" en ninguna circunstancia.

(Se podrían agregar otros defectos de menor cuantía. Pero tampoco es cosa de ensañarse con la víctima).

El lector, perplejo, se dirá, "o sea, que me quieren quitar una de las mejores herramientas de programación". Saquemos inmediatamente al estudioso de su estupor. Es cierto que le vamos (que le invitamos) a despojarle del INPUT. Pero dándole a cambio una mejor solución. O mejor, dos soluciones.

Una primera solución, incompleta y fácil es la de hacer uso de la variante del INPUT, "LINE INPUT".

Si escribimos:

```
LINE INPUT "Escribe sin miedo: ",A$
```

conseguiremos lo mismo que con INPUT pelado pero:

— Los espacios de cabeza y cola serán tenidos en cuenta como parte integrante de A$. Se esfuma el inconveniente 2.

— Los separadores —coma, punto y coma— no funcionarán como tales sino como otros tantos caracteres, con lo que también desaparece el inconveniente tercero.

— El LINE IMPUT sólo permite se le acompañe con una sola *variable alfanumérica,* con lo que el inconveniente 1 no ha lugar.

Ahí nos quedamos. Las otras pelusas permanecen. Hemos mejorado; pero nos hallamos lejos de la perfección.

Solución definitiva: el empleo de la función INPUT$(1):

Repasemos: la aparición de una instrucción del tipo:

```
20 W$=INPUT$(1)
```

hace que se pare la ejecución del programa, a la espera de que sea pulsada una tecla. Y el contenido de esa tecla, una vez pulsada, el carácter que conlleva, pasa a cargar a la variable W$, continuán-

dose la ejecución del programa en la instrucción siguiente. Recordamos que esta función.

1ro.— No precisa estar acompañada de un <CR> para que continúe la ejecución del programa (como le ocurría a la sentencia INPUT).

2do.— El carácter pulsado, no produce eco (no aparece) en pantalla.

Estudiemos el comportamiento del siguiente programa:

```
100 PRINT FNC$(10,10);
105 I$=""
110 W$=INPUT$(1)
120 IF W$=CHR$(13) THEN 200
130 PRINT W$;:I$=I$+W$
140 GOTO 110
200 ' Sigue el programa.
210 '
```

En él está el corazón de nuestro nuevo INPUT. Como en el original, el programa se parará en espera de que introduzcamos el dato que se nos antoje; y también como en aquel, el final de la introducción vendrá caracterizada por la pulsación de CR, quedando cargada la variable I$ con *todo* lo que hayamos pulsado, incluyendo posibles espacios en blanco así como «separadores» («coma» y «punto y coma»).

Tiene la desventaja que no permite (ni refleja) el retroceso. Veamos ahora este otro programa, ya más completo, en el que en la línea 60 y siguiente disponemos una *rutina de INPUT* con la misma filosofía y que, además, tiene previsto en retroceso del cursor.

```
10 DEF FNC$(H,V)=CHR$(27)+"="+CHR$(V+32)+CHR$(H+32)
20 '
30 GOTO 200
50 '
60 PRINT FNC$(10,10);:I$=""
70 W$=INPUT$(1)
80 IF W$=CHR$(13) THEN RETURN
90 IF W$<>CHR$(8) THEN 120
100 S=S-1:IF S<0 THEN S=0:GOTO 60
110 I$=LEFT$(I$,S):GOTO 130
120 I$=I$+W$:S=LEN(I$)
130 PRINT W$;
```

```
140 GOTO 70
150 '
160 '
200 H=10:V=10:GOSUB 60
210 IF I$="" THEN END
220 PRINT:PRINT I$,S:GOTO 200
```

Trate el lector de analizar por sí mismo la riqueza de la rutina 60 donde ya no queda vestigio de las limitaciones anteriormente apuntadas para el INPUT tradicional.

Aunque no es el momento de profundizar más en la cuestión, invitamos al lector a que por su cuenta y riesgo, el enorme campo que esta operatoria le abre en cuanto a:

- Fijar la naturaleza de los datos a introducir: numéricos, enteros, alfanuméricos, fechas, etc.
- Limitar la longitud del dato que se introduce.
- Jugar con la situación del cursor.
- Introducción de valores "por defecto". Etc.

PARTE SEGUNDA: MANUAL DE BASIC SISTEMATICO
ORGANIZACION Y MODO DE USO DE ESTE MANUAL

Para facilitar el estudio y/o la consulta de esta segunda parte de nuestro peregrinar por el BASIC, utilizaremos el orden alfabético sin distinción de directivas (comando, sentencia, función e incluso operador lógico, es decir, todo «vocablo BASIC») con la siguiente sistemática:

Cada comando, sentencia, etc., quedará desdoblado en:

— parte teórica (página de la izquierda) y

— parte práctica (página de la derecha).

El paralelismo entre lo teórico y lo práctico será tan completo como sea físicamente posible.

En la parte teórica se dará su expresión general, con las siguientes aclaraciones, que recomendamos sean leídas con atención y muy tenidas en cuenta.

1.— Se facilitará su traducción aproximada al castellano.

2.— Se especificará si se trata de un comando, una sentencia, una función, una variable del BASIC o un operador lógico.

3. — Se hará mención de las posibles repercusiones que tenga su inclusión en un programa que vaya a ser compilado.

4. — Si se trata de un comando, se especificará si es posible utilizarlo o no dentro de un programa.

5. — En la expresión general de las directivas, todo lo que figure entre corchetes [ ] significa que es susceptible de ser o no ser utilizado.

6. — Haremos uso de la expresión general de las siguientes abreviaturas:

Exp. Num.: Toda expresión numérica (constante, variable o fórmula numérica de cualquier tipo). Al final, con los rodeos que fueren y cualquiera que sea la complicación de la expresión, «Exp. Núm.» será simplemente un número.

Exp. Alf.: Toda expresión alfanumérica (de cadena). Mismos comentarios que en la anterior.

«Núm. Fich.»: Equivalente a «Número de Fichero». Un número entero sencillo.

«Núm. Rég.»: Equivalente a «Número de Registro». También un número entero.

«Arg.»: «Argumento». Aquellos que una función transforma.

Como aditamento práctico hemos juzgado interesante hacer notar de algún modo la importancia (o la frecuencia de uso) de cada directiva, para lo que hemos establecido tres categorías:

— Directivas de todo uso, indispensables en cualquier programa. Imposible ignorarlas. Aparecen con *doble subrayado.*

— Directivas de uso medio. Pueden ser importantes pero es posible que no se utilicen. O que se necesiten con rara frecuencia. Aparecen con *subrayado simple.*

— Directivas de uso excepcional. Puede que nunca se empleen. No se subrayan.

(Recomiéndase encarecidamente al lector primerizo, si es que se empeña en recorrer linealmente este camino, comience por el estudio de las directivas con doble subrayado; en una segunda vuelta las de subrayado simple y si, una vez pasados estos dos trances, mantiene incólume su salud mental, se lance a la conquista de las de tercera categoría).

NOTAS ADICIONALES IMPORTANTES

En algunos de los ejemplos de utilización de directivas que componen la parte práctica de este Basic Sistemático, puede que al lector le resulte a veces difícil discernir lo que *él* tiene que escribir (teclear) de lo que el propio sistema saca *automáticamente* por su cuenta.

Para paliar esta posible perplejidad hemos optado por plasmar (sólo en los casos estrictamente necesarios) en la parte derecha 3 conspicuos asteriscos 3 (***) en aquellas líneas que le corresponda teclearlas al lector. Cuando la línea esté compartida entre el sistema y el operador, será un solo asterisco lo que aparecerá a la derecha. En los casos prácticos en que es obvia la intervención del lector y la del sistema —la mayor parte— no hemos hecho uso de este artificio.

SENTENCIAS DE MANEJO DE FICHEROS: aunque en las páginas que siguen estudiamos también todas las directivas relacionadas con tratamiento de ficheros, advertimos al lector que, dada la amplitud de contenido de estas sentencias, tal vez encuentre insatisfactorio lo que aquí incluimos sobre este tema. Un desarrollo completo del manejo de ficheros podrá encontrarlo el lector en los libros «FICHEROS BASIC» y «FICHEROS BASIC: APLICACIONES» de esta misma colección.

ABS (Exp. Núm.)

Valor Absoluto.

Función. Compilable.

Da el valor absoluto de la Exp. Núm. es decir, su valor numérico positivo, tanto si la expresión es positiva como si es negativa.

AND

Significa «Y» (la conjunción copulativa de toda la vida).

Operador lógico compilable.

Como todos los operadores lógicos (a excepción del NOT), el AND tiene que estar situado entre dos proposiciones relacionales (=,>,<, etc.) o entre dos operandos. Sólo en el primer caso, la instrucción comenzará por la sentencia condicional IF.

El operador AND proporciona la *intersección* de dos conjuntos. Si estamos en presencia de proposiciones relacionales unidas por ANDs, para que se produzca resultado afirmativo (verdad) tienen que cumplirse *todas* las proposiciones de la instrucción. Si ello ocurre, la afirmación lógica da -1, y 0 si no ocurre. Basta que una sola condición no se cumpla para que se obtenga resultado 0.

Si —en lenguaje informático— nos preguntan «cuánto son 2 Y 2» no nos apresuremos a contestar «4». Porque 4 es el resultado de «2 MAS 2». En cambio «2 Y 2», o lo que es lo mismo «2 AND 2» es tanto como decir «lo que 2 tiene en común con 2». Es decir, que «2 AND 2» vale sencillamente 2. Y asimismo «12345 AND 12345» vale 12345. Y el resultado del AND aplicado a dos números cualesquiera se obtiene comparando bit a bit un número con otro, una vez que ambos números son puestos en binario. Si dos bits son «1», el resultado es «1». Cualquier otra combinación da «0». Así:

```
14 AND 37 ———————>        00001110
                    AND   00100101
                          ————————
                          00000100 ————————> 4
```

En el capítulo 2 de la primera parte, puede el lector ampliar conocimientos sobre este operador. Y si se anima a examinar con detenimiento los múltiples ejemplos de la derecha, albergamos la esperanza de que las cosas le queden claras.

```
'******  ABS  ******

PRINT ABS(-12)
 12
Ok
PRINT ABS(7);ABS(-4);ABS(-1/2);ABS(-1/3)
 7  4  .5  .333333
Ok
A=-4:B=-1:C=-2:PRINT ABS(A-C);ABS(A*B*C)
 2  8
Ok


10 ' ******   AND  ******
20 PRINT 2 AND 2; 6 AND 8; 5<3; 122>121
60 X=13:Y=2
70 FOR I=1 TO 10
80 PRINT I<X AND I>Y;
90 NEXT I:PRINT
100 '
120 FOR I=1 TO 10
130 PRINT I AND 7;
140 NEXT I:PRINT
160 PRINT 10*10; 10 AND 10; (10) AND (10)
170 PRINT X>Y AND MANUEL=MANUELA
180 PRINT Y<X AND "MANUEL">"MANUELA"
190 '
200 IF X AND Y THEN PRINT "Distintos de 0"
210 IF A=0 AND B=0 AND Y THEN PRINT "VALE"
220 '("IF Y" ES LO MISMO QUE "IF Y<>0")

RUN <CR>
 2  0  0 -1
 0  0 -1 -1 -1 -1 -1 -1 -1 -1
 1  2  3  4  5  6  7  0  1  2
 100  10  10
-1
 0
VALE
Ok
```

ASC (Exp. Alf.)

Valor ASCII.

Función. Compilable.

Proporciona el valor numérico del código ASCII que corresponde al primer carácter de la Exp. Alf.

Si la cadena-argumento (Exp. Alf.) está vacía —contiene la «cadena nula» («»), es decir, no contiene *nada*—, aparecerá un «Illegal function call» con interrupción del programa.

En la mayor parte de las aplicaciones, la Exp. Alf. es sencillamente un solo carácter. Cuando la expresión contiene más de uno, la función ASC ignora todos los que siguen al primero, único que toma en consideración. Es pues, exactamente igual escribir:

ASC(A$) que

ASC(LEFT$(A$,1)) o que

ASC(LEFT$(A$,5)) o incluso

ASC(MID$(A$,1,3)).

Los valores de esta función están comprendidos entre 0 y 255 (valores límites de un octeto).

ATN (Exp. Num.)

Contracción de «Arco Tangente», o con mayor precisión «Arco cuya tangente trigonométrica vale Exp. Num».

Función. Compilable.

Proporciona, en radianes (1 radián = 180/3.1416 grados sexagesimales), el primer ángulo cuya tangente trigonométrica es igual al argumento (Exp. Num.) de la función. Sólo se utiliza en programas técnicos (matemáticas, astronomía, topografía, etc.)

El valor de ATN lo da el sistema en simple precisión, o sea, con 6 cifras significativas.

El valor del ángulo en grados sexagesimales lo podemos obtener simplemente multiplicando el valor de ATN por 180/3.1416.

```
5  ' ******  ASC(X$)  ******
10 PRINT ASC("A"): ASC("Z")
20 PRINT ASC("ANTONIO"); ASC("ZARRAPAS")
30 A$="ALGO":  B$="ZURRON"
40 PRINT ASC(A$); ASC(B$)
50 PRINT ASC("8");
60 ASC(STR$(8)); ASC(STR$(-8))
70 PRINT ASC(MID$(B$,3,1))
80 FOR I=1 TO LEN(A$)
90 PRINT ASC(MID$(A$,I));:NEXT I
100 PRINT: PRINT ASC(111)

RUN <CR>
 65  90
 65  90
 65  90
 56  32  45
 82
 65  76  71  79
Type mismatch
Ok
PRINT ASC(JAMON$)
Illegal function call
Ok


1 ' ******  ATN(X)  ******
2 PRINT ATN(1); ATN(2)
10 A=1.22: T=TAN(A): PRINT ATN(T)
40 PI=3.1416: CONVER=180/PI
60 PRINT ATN(1)*CONVER; ATN(1000)*CONVER;
70 PRINT ATN(-1)*CONVER
80 PRINT ATN(10^30)

RUN <CR>
 .785398  1.10715
 1.22
 45  89.9427  -45
 1.57079
Ok
```

AUTO (Nro. de línea de comienzo [,Incremento])

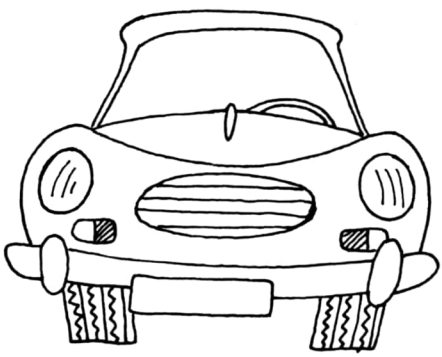
Prefijo de «Autoprogramación».

Comando. Irrelevante en compilado.

Utilizable (aunque sin sentido o con un sentido asaz jocoso) dentro de un programa.

Genera un número de línea cada vez que se pulsa CR. La primera línea será la que figura en primer lugar tras el AUTO.

Los saltos numéricos entre líneas serán los señalados por el incremento. El valor por defecto es 10, tanto para la línea de comienzo como para el incremento.



Si la línea generada resultase estar ya utilizada, aparecerá, con ella, un asterisco (*) para advertir al programador de que corre el riesgo de fulminar una línea que puede ser valiosa, cosa que sucederá si escribe lo más mínimo.

Si, ante una línea con asterisco, se limita a pulsar CR, el sistema entiende que quiere conservar la línea existente, pasando a ofrecer la siguiente.

Se suele utilizar en programación auxiliar, no en la principal donde con gran frecuencia hay que estar intercalando líneas. Se termina pulsando CONTROL-C (como el abortado de un programa en curso de ejecución).

```
AUTO <CR>
10 'PRUEBA DE AUTO. LOS NUMEROS DE LINEA
20 'VAN SALIENDO AUTOMATICAMENTE, CON CADA PULSACION DE CR.
30 'NOSOTROS VAMOS ESCRIBIENDO
40 'TODAS ESTAS COSAS, Y LOS NUMEROS DE LINEA SALEN
50 'ELLAS SOLAS, SIN INTERRUPCION, DE 10 EN 10
60 'MIENTRAS NO SE PULSE CONTROL-C
70 'COSA QUE VAMOS A HACER AHORA, CUANDO APAREZCA LA LINEA 80
80^C
Ok

LIST <CR>
10 'PRUEBA DE AUTO. LOS NUMEROS DE LINEA
20 'VAN SALIENDO AUTOMATICAMENTE, CON CADA PULSACION DE CR
30 'NOSOTROS VAMOS ESCRIBIENDO
40 'TODAS ESTAS COSAS, Y LOS NUMEROS DE LINEA SALEN
50 'ELLAS SOLAS, SIN INTERRUPCION, DE 10 EN 10
60 'MIENTRAS NO SE PULSE CONTROL-C
70 'COSA QUE VAMOS A HACER AHORA, CUANDO APAREZCA LA LINEA 80
Ok

AUTO <CR>
10*
20*
30*
40*'MODIFICAMOS ESTA LINEA (ESCRIBIMOS ESTO) **********
50*'Y ESTA OTRA. **********
Ok

LIST <CR>
10 'PRUEBA DE AUTO. LAS LINEAS DE PROGRAMA
20 'VAN SALIENDO AUTOMATICAMENTE, CON CADA PULSACION DE CR
30 'NOSOTROS VAMOS ESCRIBIENDO
40 'MODIFICAMOS ESTA LINEA (ESCRIBIMOS ESTO) **********
50 'Y ESTA OTRA. **********
60 'MIENTRAS NO SE PULSE CONTROL-C
70 'COSA QUE VAMOS A HACER AHORA, CUANDO APAREZCA LA LINEA 80
Ok
```

CALL Nombre de Programa (Lista de Argumentos)

Call = «Llamada».

Sentencia. Compilable (ver reglas operatorias en Compilación, Anexo 2).

Esta sentencia es una de las pocas que, formando parte del Basic, llama y ejecuta otro programa (rutina) que se encuentra presente en memoria y que ha sido desarrollado en otro lenguaje (Ensamblador, Fortran, Pascal, etc.).

La lista de argumentos contiene los parámetros que han de pasar a la rutina. Esos argumentos tienen que ir separados por comas.

No importa el nombre que se dé a los argumentos cuando pasan en forma de variables; es imprescindible sin embargo, que sean del tipo exigido por el programa «llamado».

El «Nombre de Programa», cuando se trabaja en intérprete, contiene la dirección de comienzo de la rutina. En compilado, esta dirección es designada directamente por el Linker.

CDBL (Exp. Num.)

Abreviatura de «CONVERT DOUBLE». Función compilable.

Convierte el número que corresponda a Exp. Num. en un número de doble precisión. Función poco utilizada y algo peligrosa, pues al efectuar la conversión produce una especie de «anti-redondeos» como se ve en los ejemplos, que pueden llevarnos a un disgusto serio. Recomiéndase, cuando se desee que una variable tenga doble precisión, se defina como tal (agregando #) o con la función específica DEFBL (véase).

```
10 '****** CALL ******
20 SUMA=&HA400
30 A=11
40 B=14
45 '
50 CALL SUMA(A,B,S)
60 'En la rutina SUMA se produce
65 'nada menos que la suma de A y B.
75 PRINT S
80 PEPO=&HA800
85 '
90 CALL PEPO(A,B,S,T)
100 PRINT A;B;S;T
110 'Se supone que esta segunda CALL
120 'produce una transformaciòn tipo
125 'mangulino-matematica de las
140 'variables-argumento en cuestion.

RUN <CR>
 25
 11   14   25   1198764.23234555
Ok



10 '*****  CDBL(Exp.Num.)  *****
20 '
30 X=1/2:Y=1/3:Z=1/7
40 X#=1#/2:Y#=1#/3:Z#=1#/7
50 PRINT X;Y;Z
60 PRINT CDBL(X);CDBL(Y);CDBL(Z)
70 PRINT X#;Y#;Z#

RUN <CR>

 .5   .333333   .142857
 .5   .3333333432674408   .1428571492433548
 .5   .3333333333333333   .1428571428571429
Ok
```

CHAIN [MERGE] Nombre del programa (,[Número de línea] [,ALL] [,DELETE Rango]

Chain = «Encadenamiento», «Enlace con».

Sentencia. El compilador no admite las alternativas (entre corchetes) que aparecen en la expresión general.

Esta sentencia, formando parte de una línea del programa en el que estamos, pone en ejecución otro programa totalmente independiente cuyo nombre es el que figura junto a CHAIN. El uso de CHAIN permite segmentar y estructurar la aplicación a la medida de nuestros deseos y objetivos. Toda una familia de programas pueden circular con toda soltura de la memoria externa a la interna sin ninguna limitación.

Con la opción «Nro. de línea» la ejecución del programa llamado da comienzo en el número de línea especificado. Si se omite esta opción, se ejecuta el programa encadenado desde su comienzo. La opción ALL (todos) hace que todas las variables con sus valores en el momento del cambio de programa, se transmitan al programa «encadenado».

Si no se utiliza la opción ALL, hay que recurrir a la sentencia COMMON para pasar los valores de las variables que se desee estén presentes en el nuevo programa (véase COMMON para mayor claridad).

Si se incluye la opción MERGE (mezclar) ello permite que el nuevo programa entre en memoria solapándose con el original y desapareciendo tan pronto como hagamos uso de la opción DELETE (borrar). Por útiles que estas opciones parezcan, al no ser utilizables con el compilador, no insistimos en ellas.

Importante anotar que la sentencia CHAIN (en compilado o en intérprete) no cierra los ficheros que pudieran estar abiertos en el momento de la transferencia.

```
10 ' *****  CHAIN  *****
20 ' CUANDO ESTE PROGRAMA LLEGUE A LA
25 ' LINEA 70, DESAPARECERA, BARRIDO
60 ' POR EL DENOMINADO "ENCADENA"
65 ' QUE EMPEZARA A EJECUTARSE.
70 CHAIN "ENCADENA"


10 'ESTE ES OTRO PROGRAMA DISTINTO
12 'QUE ESTA SALVADO CON NOMBRE "PROGR1"
16 '
20 A=200:B=500
30 PRINT "PROGR1:";A;B
40 IF A=0 THEN LIST
45 'SI A=0 LISTA EL PROGRAMA Y TERMINA
50 CHAIN MERGE "PROGR2",60,ALL


35 'PROGRAMA SALVADO COMO "PROGR2".
55 PRINT "ESTA LINEA NUNCA SE EJECUTA"
60 PRINT "PROGR2:";A,B
70 PRINT "HA FUNCIONADO EL 'ALL' DE PROGR1"
80 CHAIN "PROGR1",30

RUN "PROGR1" <CR>
PROGR1: 200  500
PROGR2: 200  500
HA FUNCIONADO EL 'ALL' DE PROGR1
PROGR1: 0  0
10 'ESTE ES OTRO PROGRAMA DISTINTO
12 'QUE ESTA SALVADO CON NOMBRE "PROGR1"
16 '
20 A=200:B:500
30 PRINT "PROGR1:";A;B
40 IF A=0 THEN LIST
45 'SI A=0 LISTA EL PROGRAMA Y TERMINA
50 CHAIN MERGE "PROGR2",60,ALL
Ok
```

CHR$ (Exp. Num.)

Primeras consonantes de «Character» (=carácter).

Función compilable.

Es algo así como el reverso de la moneda de la función ASC. Da el carácter que corresponde al valor de la Exp. Num. Naturalmente ese valor tiene que estar entre los límites 0-255 si no queremos que nos salte el consabido «Illegal Function Call».

De los 256 caracteres diferentes que se pueden formar con un octeto, unos son invisibles (del 0 al 31) y otros visibles (del 32 en adelante). Entre los invisibles podemos citar como ejemplo:

CHR$(7) BELL, aviso acústico, pitido breve estridente.

CHR$(9) TAB, tabulador.

CHR$(10) LF, salto de línea.

CHR$(12) FF, salto de página en impresora.

CHR$(13) CR, Retorno de carro.

CHR$(27) ESC, Escape, carácter que modifica al o a los que le sigan.

Y entre los visibles:

CHR$(48) «0».

CHR$(65) «A».

CHR$(97) «a».

Etc.

(Véase Anexo 3 en el que se dan los caracteres que corresponden a cada ASCII).

CINT (Exp. Num.)

Contracción de «Convert Integer» (conversión en entero).

Función compilable. Convierte al valor de Exp. Num. en un «entero» (entre -32768 y 32767) redondeando por exceso según el valor de la parte decimal que pueda tener. No confundir con la función INT (ver ésta y ejemplos en ambas).

Poco utilizada, de la familia de CSNG y CDBL aunque sin los peligros de esta última (véase).

```
10 ' ****** CHR$(X) ******
20 PRINT CHR$(90)
30 FOR I=1 TO 20: PRINT CHR$(45);:NEXT
60 PRINT:PRINT CHR$(49);CHR$(10);CHR$(50);
70 PRINT CHR$(10);CHR$(51)
80 FOR I=1 TO 100
90 PRINT "INESITA";CHR$(13);
100 NEXT I:PRINT
120 PRINT CHR$(ASC("G")); ASC(CHR$(100))
140 FOR I=1 TO 10
150 PRINT CHR$(47+I);
160 NEXT:PRINT
180 DATA 68,65,66,65,32,68,65,66,65,68,65
190 FOR A=1 TO 11
200 READ X:PRINT CHR$(X);:NEXT:PRINT
210 INPUT "CODIGO (0=FIN): ",COD
215 IF COD=0 THEN END
220 PRINT CHR$(COD):GOTO 210

RUN <CR>
Z
--------------------
1
 2
  3
INESITA
G 100
0123456789
DABA DABADA
CODIGO (0=FIN): 68
D
CODIGO (0=FIN): 0
Ok

A=12345:PRINT CINT(A);CINT(A/2)
 12345  6172
Ok
PRINT CINT(111.5);CINT(111.4);CINT(111.6)
 112  111  112
Ok
```

CLEAR[,[Expresión 1] [,Expresión 2]

Clear = «Vaciar», «Limpiar».

Sentencia. Bastante peligrosa, por cierto.

El compilador no tiene inconveniente en utilizarla siempre que tanto la Expresión 1 como la Expresión 2 sean enteros (%).

Esta demoledora sentencia hace (más bien deshace) lo siguiente:

— Pone a cero todas las variables numéricas.

— Vacía de contenido todas las variables alfanuméricas.

— Cierra todos los ficheros que puedan estar abiertos.

— Libera los buffers de los discos/diskettes.

En resumidas cuentas, a excepción del programa desnudo, produce un total exterminio de cuanto en memoria interna haya. Esto se traduce por una desaparición de todos los datos que pudiese haber en el «Area de Datos» de la memoria interna.

Lo único que escapa al exterminio (lo único que no se borra del área de datos) son las variables del BASIC relacionadas con el tratamiento de errores: ERR y ERL.

La opción «Expresión 1», si es que se utiliza, establece la dirección más alta de memoria disponible para el intérprete BASIC.

La opción «Expresión 2» establece el espacio del stack. El valor por defecto de éste es de 256 bytes o 1/8 de la memoria disponible (el menor de estos dos valores). Dado el nivel en el que nos movemos, rogamos al lector novel no profundice en algo que seguramente no va a emplear, aun cuando acometa muy serios y amplios programas.

```
10 '****** CLEAR ******
20 '
30 DIM VEC(255),VEC$(255)
35 PRINT
40 '
50 FOR X=0 TO 255
60 VEC(X)=X
70 VEC$(X)=CHR$(X)
80 NEXT
90 '
100 FOR I=1 TO 10
110 PRINT VEC(I);
120 NEXT I
130 PRINT: PRINT
140 '
150 FOR I=97 TO 122
160 IF VEC$(I)<>"" THEN 180
170 PRINT "X";:GOTO 190
180 PRINT VEC$(I);
190 NEXT
200 PRINT: PRINT
210 '
220 '
230 IF VEC$(33)="" THEN END
240 '
250 CLEAR
260 '
270 GOTO 100

RUN <CR>

 1  2  3  4  5  6  7  8  9  10

abcdefghijklmnopqrstuvwxyz

 0  0  0  0  0  0  0  0  0  0
Suscript out of range in 160
Ok
```

CLOSE ([#] Nro. Fichero [,[#] Nro Fichero...]]

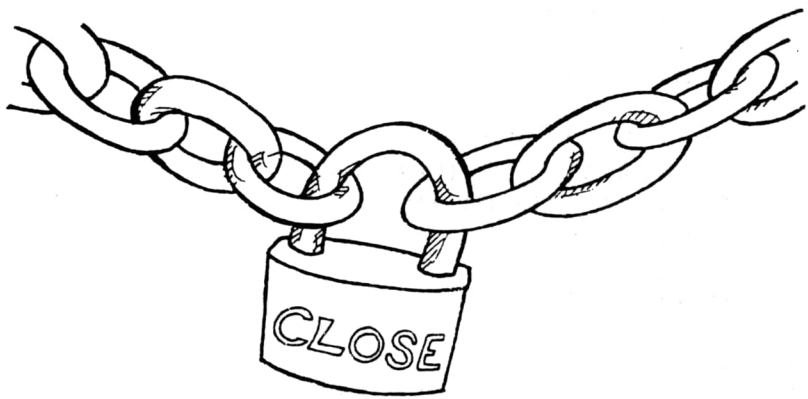
Close (=cerrar).

Sentencia específica de manejo de ficheros.

Compilable sin restricciones.

Si no se utilizan las opciones, se cierran *todos* los ficheros que pudieran estar abiertos. Si se especifica(n) número(s) de fichero(s), se cierran los indicados, deshaciéndose la conexión entre Nro. y Nombre de Fichero que existía cuando éste se abrió, con lo que el Nro. puede ser reutilizado en una nueva apertura.

El signo # (Nro.) puede ser omitido.



Cuando se está grabando un fichero secuencial, éste no queda incorporado al directorio del disco hasta que no se efectúa un CLOSE del mismo. Esto es importante porque un olvido de esta sentencia puede dejar sin efecto un laborioso trabajo.

Las sentencias END, NEW y CLEAR producen un cierre automático de todos los ficheros. No así (y por fortuna), la sentencia STOP.

Si hacemos un CLOSE C, y no había ningún fichero abierto por el canal C, no se produce error. Un CLOSE a secas cierra todos los ficheros y tampoco acarrea ningún error si no hubiese ninguno abierto.

(Para una información más específica y práctica de esta sentencia, remitimos al lector al volumen «FICHEROS BASIC» dentro de esta misma colección.)

```
10 ' ******  CLOSE  ******
20 ' Ver tambien OPEN, FIELD, GET, etc.
30 CLOSE:CLOSE 3
40 'Nada hay que cerrar (no salta error).

RUN <CR>
Ok

CLOSE <CR>
Ok

CLOSE 1,2,3 <CR>
Ok

10 OPEN "R",1,"A:COSAS",24
20 FIELD 1, 24 AS X$
30 GET 1: A$=X$
40 CLOSE 1
50 OPEN "I", 1, "B:ENDECHAS"
60 INPUT #1, B$
70 PRINT A$: PRINT B$
80 OPEN "R", 1, "A:COSAS"

RUN <CR>
ALICATES GRANDES
ANOCHE CUANDO DORMIA
File already open in 80
Ok

RUN 80 <CR>
Ok

GOTO 80 <CR>
File already open in 80
Ok

CLOSE:GOTO 80 <CR>
Ok
```

COMMON Lista de Variables

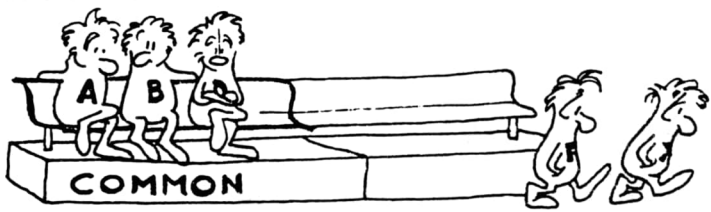
Common = «Común»

Sentencia específica para el paso de variables entre programas.

Compilable (véanse comentarios más abajo).

Si no existe en el programa ninguna sentencia CHAIN, la COMMON está de más (no sirve para nada). Se recomienda el uso del COMMON al comienzo del programa (véase Anexo 2).

Los vectores y matrices (o variables indexadas) pueden ser incluidas en una sentencia COMMON sin más que adosarles unos paréntesis vacíos ( ). Una variable de este tipo, incluida en un COMMON debe estar previamente dimensionada (véase sentencia DIM).



El compilador contempla gustoso la sentencia COMMON con la única exigencia adicional de que esta sentencia ha de aparecer en el programa *antes de* ninguna sentencia llamada «ejecutable». Y, como bien diría Perogrullo son ejecutables todas, menos las que no lo son, a saber:

- Las 4 DEF: DEFDBL, DEFSNG, DEFINT, DEFSTR.
- DIM
- OPTION BASE
- REM
- %INCLUDE
- COMMON (naturalmente).

Tomemos buena nota, por último, que las sentencias COMMON tienen que ser *idénticas* en el programa enlazante (por CHAIN) y en el enlazado, tanto en el número de variables como el orden en que éstas están situadas en la sentencia COMMON.

El mejor modo de asegurar inexactitudes fatídicas en este sentido, es ir al empleo de la original y solitaria sentencia INCLUDE cuyo uso se explica en el Anexo 2.

```
10 ' ***** COMMON *****
20 'Este programa esta salvado como "B:PEPE"
30 'Es decir, como PEPE en el drive o disco B
50 IF A$<>"" THEN 110
60 'Si no ponemos la instrucción anterior y trabajamos
70 'en interprete (sin compilar), nos apareceria una
80 'horrorosa "Duplicate definition in 50".
90 '
100 DIM A(20)
110 COMMON A(), A$, PP$
120 PRINT A(1); A(17); B; PP$; " "; A$; " "; B$
130 A(17)=1001
140 A$="ZAMBOMBA": PP$="ATIZA, GOROSTIZA"
150 B$="CASCARAS": B=333
160 A(1)=A(1)+1
170 IF A(1)>2 THEN END
180 CHAIN "B:PACO"


10 'Este otro programa esta salvado como "B:PACO"
20 'O sea, en el disco B con el nombre de PACO.
30 '
40 'Si el programa no va a ser compilado
50 'no debemos volver a DIMensionar los vectores
60 'lo cual implica que a este programa no vamos
70 'a acceder directamente, sino siempre desde
80 'el programa "PEPE", el cual trae ya el DIM
90 'preparado. Y en el caso en que vaya a ser
100 'compilado, es preciso utilizar el DIM en cada
110 'programa para que el COMMON funcione.
130 COMMON A(), A$, PP$
140 PRINT A(1); A(17); PP$; " "; A$; " "; B$
150 A(1)=A(1)+1: A(17)=123456
170 CHAIN "B:PEPE"

RUN "B:PEPE" <CR>
 0  0
 1  1001  0 ATIZA, GOROSTIZA ZAMBOMBA
 2  123456  0 ATIZA, GOROSTIZA ZAMBOMBA
Ok
```

CONT

De CONTINUE (Continuar, Reanudar)

Comando. No puede formar parte de un programa. Nada que hacer por tanto en compilación.

Hace proseguir la ejecución de un programa que se ha interrumpido, bien por una sentencia STOP dentro del programa o por un abortado del mismo mediante un CONTROL-C. Típico comando de intérprete. Su uso principal se da en la depuración de un programa en el que se intercalan los necesarios STOPs para ir siguiendo la pista de los valores de las variables y, en general, de la marcha del programa.

Después de las verificaciones en modo directo, un CONT <CR> reanuda la ejecución del programa exactamente en el punto en que se interrumpió, incluso aunque en la fase anterior se hayan asignado valores directos a las variables.

Si se ha efectuado un EDIT, no se puede ejercitar el CONT. Se ha producido un CLEAR automático por lo que un intento de CONT dará lugar a la aparición de un mensaje así:

Can't continue (= no me es posible continuar)

COS (Exp. Num.)

De Cosinus (Coseno, función trigonométrica).

Función, compilable. Proporciona el valor (en simple precisión: 6 cifras significativas) del conseno del ángulo. Exp. Num. La función considera en radianes al argumento. Son aplicables por tanto los comentarios incluidos en la función ATN a la que nos remitimos para más información.

```
10 '****** CONT ******
15 '
20 W$=W$+"A"
30 IF LEN(W$)<128 THEN 20
40 STOP
50 PRINT "COMO VES, HE PODIDO CONTINUAR."
RUN <CR>
Break at line 40
Ok
PRINT LEN(W$)
 128
Ok
CONT <CR>
COMO VES, HE PODIDO CONTINUAR.
Ok

10 X=X+7:PRINT X: '(OTRO PROGRAMITA)
30 STOP
40 X=X+1:PRINT X:STOP:GOTO 10

RUN <CR>
 7
Break at line 30
Ok
CONT <CR>
 8
Break at line 40
Ok


'******  COS  ******
PRINT COS(0) <CR>
 1
Ok

FOR I=1 TO 4:PRINT COS(I);" ":NEXT <CR>
 .540302  -.416147  -.989993  -.653644
Ok
```

CSNG (Exp. Num.)

De «Convert single» (Conversión a simple precisión).

Función compilable y de escaso empleo.

Convierte a la Exp. Num. en un número de simple precisión (4 bytes). Véanse comentarios en las funciones CDBL y CINT.

CVI (Exp. Alf.)
CVS (Exp. Alf.)
CVD (Exp. Alf.)

Abreviaturas de «Convert Integer», «Convert single» y «Convert Double» respectivamente (conversión en Entero, en Simple y en Doble precisión).

Funciones compilables.

Estas funciones están íntimamente relacionadas con las variables numéricas de campo de un fichero de acceso directo.

De forma general, toman los valores del argumento (Exp. Alf.) y lo *desempaquetan*, es decir, lo traducen al número correspondiente. La longitud del argumento debe ser;

- De 2 bytes para CVI.
- De 4 bytes para CVS.
- de 8 bytes para CVD.

El rango de precisión, por tanto, de los números así obtenidos es:

- Entero para CVI.
- Simple precisión para CVS.
- Doble precisión para CVD.

(Para profundizar en el empleo práctico de estas funciones, véase la repetidamente mencionada nota respecto a instrucciones de manejo de ficheros del comienzo de esta segunda parte).

```
' ***** CSNG(X) *****
A#=12345678#: A=CSNG(A#): PRINT A#,A <CR>
 12345678      1.23457E+07
Ok


10 ' ***** CVI  CVS  CVD *****
20 '
30 A%=CVI("kk"):PRINT A%
40 A=CVS("kkkk"):PRINT A
50 A#=CVD("kkkkkkkk"):PRINT A#
60 ' Lo anterior es solo un divertimento, para senalar que
70 ' estas funciones realmente traducen cualquier cadena
80 ' de 2, 4 u 8 bytes segun se trate de CVI, CVS o CVD.
90 '
100 A%=12345: A=123456!: A#=1234567890123458
110 A1$=MKI$(A%): A2$=MKS$(A): A3$=MKD$(A#)
120 PRINT A1$;A2$;A3$
125 '
130 'De nuevo vemos la inutilidad de conocer
140 'los contenidos de las cadenas MKI$ etc.
150 'El uso, pues, de estas funciones solo
160 'tienen sentido cuando se emplean conjuntamente.
170 '
180 PRINT CVI(A1$): PRINT CVS(A2$): PRINT CVD(A3$)
185 '
190 'Con el uso de las MKI$ etc., empaquetamos
200 'para grabar correctamente en ficheros.
210 'Y con el uso de las CVI etc., desempaquetamos
220 'para traducir a numerico lo alfanumerico.

RUN <CR>
 27499
 4.38503E-07
 4.385031905828738D-07
90 q~ 'r> y'/
 12345
 123456
 123456789012345
Ok
```

DATA Lista de constantes numéricas y/o de cadena

Del latín, «Datos».

Sentencia compilable.

Esta sentencia no es ejecutable (no sirve para nada por sí sola) pues funciona sólo bajo las órdenes de la sentencia READ. Puede ir en cualquier parte del programa.

La lista de constantes puede ser tan larga como lo permita la línea, hasta completar los 254 bytes. Los datos han de ir separados por comas. Atención: deben ser *constantes.* Por ejemplo, si uno de los DATA es «A», se supone que se trata de la constante alfanumérica «A» y no de la variable numérica «A». Puede haber tantas líneas DATA en el programa como se quiera, sin más restricción, como siempre, que la capacidad de memoria.

Las constantes que acompañan a los DATA permanecen estáticas, esperando el advenimiento de un READ que las haga ser poseídas por la variable que le acompaña, cobrando así vida dentro del programa.

Si no hay sentencia RESTORE que haga cambiar el rumbo de las lecturas, las constantes de los DATA son leídas por estricto orden: orden creciente de líneas (cualesquiera que éstas sean, estén al principio, en medio o al final del programa), y dentro de cada línea, orden secuencial de las constantes incorporadas.

Si las constantes son numéricas, éstas pueden ser de cualquier precisión y tener cualquier formato (coma fija o coma flotante). Si son alfanuméricas pueden ir con o sin comillas. En el primer caso, los espacios que pueda haber al principio o final de ellas no entran a formar parte de la variable a las que los READ las incorpora. Lo mismo sucede con las comas que en tal caso actúan de verdaderos separadores de datos. Si, en cambio, las constantes alfanuméricas se escriben entrecomilladas, los espacios iniciales o finales y las comas que pueda haber son incluidos como otros caracteres cualesquiera.

Hacemos destacar, por último, que la variable que lleva el READ debe ser del mismo tipo que la constante leída; de otro modo surgirá un abominable «Type Mismathc».

(Véase primera parte de este volumen así como sentencias READ y RESTORE dentro de esta segunda parte).

```
10 ' ******  DATA  ******
12 DATA PEDRO,ANDRES,SANTIAGO,JUAN,TOMAS,SANTIAGO
15 DIM APOST$(12),AUSTR$(5),NOTAS(80)
30 FOR I=1 TO 12
40 DATA FELIPE,BARTOLOME,MATEO,SIMON,TADEO,JUDAS
50 READ APOST$(I)
70 NEXT I
80 DATA CARLOS I,FELIPE II,FELIPE III,FELIPE IV
90 FOR I=1 TO 5
100 DATA CARLOS II,4,6,2,9,9,5,6,5,6,8,4,8,6,6
110 READ AUSTR$(I)
120 NEXT I
140 DATA 7,5,5,6,4,0,3,9,9,6,7,3,7,7,6,5,5,10,9
160 '
170 FOR I=1 TO 80
180 DATA 7,7,5,3,7,5,9,2,5,6,1,4,6,7,8,7,7,6,7,8
190 READ NOTAS(I)
200 NEXT I
210 '
230 FOR I=1 TO 5
240 PRINT AUSTR$(I)
250 DATA 6,4,8,9,5,5,3,7,8,7
260 NEXT I
270 '
290 FOR I=1 TO 7:PRINT APOST$(I);" ";:NEXT:PRINT:PRINT
300 FOR I=50 TO 60:PRINT NOTAS(I);:NEXT
320 'Sirve para demostrar que las instrucciones DATA
330 'pueden situarse en cualquier parte del programa.
340 DATA 5,5,6,7,5,3,8,9,0,10,3,7,7,5,4,4,9

RUN <CR>
CARLOS I
FELIPE II
FELIPE III
FELIPE IV
CARLOS II
PEDRO ANDRES SANTIAGO JUAN TOMAS SANTIAGO FELIPE

 7  6  7  8  6  4  8  9  5  5  3
Ok
```

DEF FN Nombre (Lista de Parámetros) = Definición función

Viene de «Definition of function» cuya traducción es obvia.

Sentencia compilable y no ejecutable en modo directo.

Con esta directiva se define una función numérica o alfanumérica que queda a disposición del programador que la ha definido para su utilización a lo largo del programa.

El «Nombre» de la función puede ser cualquier variable numérica o de cadena con las mismas condiciones a las que éstas están sometidas.

En la lista de parámetros van éstos, separados por comas, con nomenclatura de variables. En rigor no lo son, sino simples referencias para poder definir la función. Tan no lo son, que no afectan para nada a las variables que realmente tenga el programa con el mismo nombre.

La expresión con la que se define la función (derecha del signo «=») está limitada a una sola línea.

Un nombre de variable utilizado en la definición de la función puede aparecer o no en la lista de parámetros. Si está en la lista, al ser utilizada la función en el programa, el valor concreto que en él se dé al parámetro en un momento dado, entra en la función y actúa con arreglo a ella. Y si no está en la lista, tomará el valor que la variable tenga en el momento de su uso sin más. Esto puede parecer algo confuso. Véanse los ejemplos adjuntos donde confiamos queden claras sus excelencias y ventajas. Y también el capítulo 6 de la primera parte.

Es importante resaltar:

- El orden de la lista de variables en el *uso* de la función ha de ser el mismo que el de la *definición.*
- El nombre de la función debe ser idéntico en la definición y en el empleo.
- Obviamente, la definición de la función tiene que ser previa a su utilización.

```
10 ' ****** DEF FN ******
20 'SUPONGAMOS QUE EN UN PROGRAMA SE UTILIZA
30 'CON FRECUENCIA EL TERCER CARACTER POR LA
40 'DERECHA DE UNA CADENA DE LONGITUD VARIABLE.
60 DEF FNJ$(A$)=LEFT$(RIGHT$(A$,3),1)
80 INPUT "Cadena (F=Fin): ",C$:IF C$="F" THEN END
100 IF LEN(C$)<3 THEN 80
110 PRINT LEFT$(RIGHT$(C$,3),1);FNJ$(C$):GOTO 80

RUN <CR>
Cadena (F=Fin): Barbara
aa
Cadena (F=Fin): ROSA
OO
Cadena (F=Fin): F
Ok

10 'SI TENEMOS QUE EMPLEAR A MENUDO UNA FORMULA
20 DEF FNC(R,G,H)=3.14*R*G + 6.28*R*H + 3.14*R^2
30 INPUT "RADIO, GENERATRIZ, ALTURA: ",A,B,C
40 PRINT FNC(A,B,C);FNC(A/2,B/4,C/8)

RUN <CR>
RADIO, GENERATRIZ, ALTURA: 22,61,13
 7529.72  1018.93
Ok

10 'SI QUEREMOS RELLENAR CON CEROS POR LA IZQUIERDA
20 DEF FNN$(A#,B)=RIGHT$(STRING$(80,48)+MID$(STR$(A#),2),B)
30 INPUT X#:IF X#=0 THEN END ELSE PRINT FNN$(X#,8):GOTO 30

RUN <CR>
?12
00000012
?154
00000154
?9999
00009999
?0
Ok
```

DEFINT (Rango de letras)
DEFSNG Id.
DEFDBL Id.
DEFSTR Id.

Traducibles por «Definimos como enteras, como de simple precisión, como de doble precisión, como variables de cadena», respectivamente.

Sentencias compilables. Con DEFINT H por ejemplo, definimos que todas las variables que empiecen por H son variables enteras. No hará falta, pues, el aditamento «%» que era propio de estas variables.

Si la definición es DEFSTR Q-U, estamos obligando al programa a que toda variable que empiece por Q, R, S, T ó U, sin el típico signo «$» de las cadenas, se comporte como variable alfanumérica, como si tuviera, sin que lo tenga, el áureo signo. Y si ponemos DEFDBL A-Z, todas las variables serán de doble precisión. Esto es general, pero queda subordinado al posible empleo del pegote ($, %, ¡, #) que prevalece sobre cualquier definición previa de las variables.

La definición de letras alcanza sólo a la *primera* (no se puede escribir DEFSNG AB-AF, por ejemplo). Cuando una variable (su inicial) no ha sido definida, su «tipo» por defecto es el de variable numérica de simple precisión.

DELETE Nro. de Línea-Nro. de Línea

Delete = Borrar.

Comando. Irrelevante en compilación.

Sirve para borrar una parte del programa que hay en memoria. Cuando se ejecuta el comando, desaparecen del programa y de la memoria, todas las líneas comprendidas entre los guiones (incluidas esas mismas). Es necesario que *existan* en el programa ambas líneas extremas, aunque, a despecho de esto, la línea-comienzo de programa se puede omitir. Comando muy útil en programación, especialmente en fusión (MERGE) de programas.

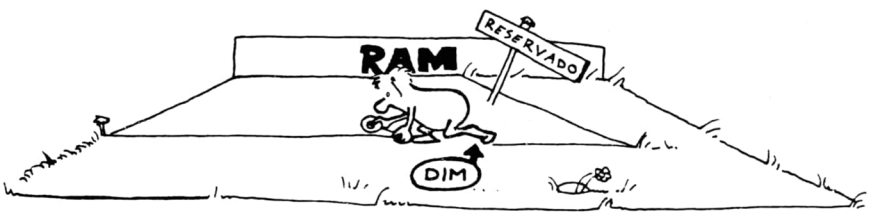
```
'******  DEFINT  DEFSNG  DEFDBL  DEFSTR  ******
PRINT 77777/3 <CR>
 25925.7
DEFINT A: A=77777!/3: PRINT A <CR>
 25926
DEFDBL A: A=77777000/3: PRINT A <CR>
 25925666.66666667
Ok
DEFINT A-I: DEFDBL J-N: EMILIA=1/3: PRINT EMILIA <CR>
 0
Ok
DEFSTR S: S=4 <CR>
Type mismatch
S="ESTRELLAS": STAR="LUPE": PRINT S;" ";STAR <CR>
ESTRELLAS LUPE
INPUT "NUMERO: ",F:PRINT F <CR>
NUMERO: 33333
Overflow
INPUT "NUMERO: ",N:PRINT N;:CC=N <CR>
NUMERO: 9999999999
 9999999999 Overflow
INPUT "NUMERO: ",X:PRINT X <CR>
NUMERO: 9999999999
 1E+10
S1=MKI$(F):S2=MKS$(X):S3=MKD$(N) <CR>
Ok
S8=MKI$(N)
Overflow

' ******   DELETE   ******
10 A=2:B=7:C=LOG(A*B/2)
20 X$="PATO DONALD"
30 REM
40 REM EMILIOCO Y RAMONZA

DELETE -30
LIST
40 REM EMILIOCO Y RAMONZA
Ok
```

DIM Lista de Variables Indexadas

De «Dimensionado de».

Compilable. Típica sentencia incorporable al comienzo del programa. Gracias a ella se pueden utilizar vectores dimensionados con subíndices tan altos como la capacidad de memoria lo permita.

El tipo de variable a dimensionar puede ser cualquiera (alfanumérica, numérica entera, etc.). Si no se dimensiona una variable y se utiliza como vector o matriz, el subíndice máximo que permite el BASIC sin producir error es el 10. Por cuestiones de orden y de un posible ahorro de memoria, recomiéndase su uso en *todos* los casos en que vaya a hacerse uso de este tipo de variables.



En cualquier caso, si se le asigna a una variable un subíndice superior al especificado en la sentencia DIM (o superior a 10 si no ha sido dimensionada) aparecerá en pantalla la acusación de «Subscript out of range».

El valor mínimo de subíndice de todas las variables uni o multidimensionales (a menos que se especifique de otra manera mediante el uso de la sentencia OPTION BASE) es de *0*. Con ello, el número de elementos de que una variable de esta clase dispone, es de N+1 (siendo N el valor de la dimensión o 10 en su defecto).

El valor inicial (inmediatamente después del DIM) de todos los elementos es —como en las variables simples— de 0 para las numéricas y de cadena vacía («») para las alfanuméricas.

Si se intenta volver a dimensionar una variable indexada, salta el error («Duplicate definition»).

Si se va a emplear la sentencia COMMON, ésta debe ir después de la DIM (véase COMMON).

```
10 '******   DIM   *******
20 DIM NOMBRE$(20),EDAD(20)
30 FOR A=17 TO 20
40 INPUT "Nombre: ",NOMBRE$(A)
50 INPUT "Edad: ",EDAD(A)
60 NEXT A
70 FOR A=17 TO 20
80 PRINT NOMBRE$(A);" tiene";EDAD(A);"primaveras."
90 NEXT A

RUN <CR>
Nombre: Cosme
Edad: 47
Nombre: Requlciperio
Edad: 58
Nombre: Rogelia
Edad: 81
Nombre: Ingrid
Edad: 19
Cosme tiene 47 primaveras.
Requlciperio tiene 58 primaveras.
Rogelia tiene 81 primaveras.
Ingrid tiene 19 primaveras.
Ok

10 DIM A$(17,10),CONTA(17)
20 DATA GALICIA,ANDALUCIA,CANARIAS,BALEARES,PAIS VASCO
30 DATA MADRID,CASTILLA-LEON,CATALUNYA,ARAGON,ASTURIAS
40 DATA CANTABRIA,RIOJA,NAVARRA,CASTILLA-LA MANCHA
50 DATA VALENCIA,MURCIA,EXTREMADURA
60 FOR I=1 TO 17:READ A$(I,0):NEXT
70 INPUT "PROVINCIA (0 FIN): ",PR$:IF PR$="FIN" THEN 200
90 FOR I=1 TO 17: PRINT I,A$(I,0):NEXT I
100 INPUT "INDICA NRO. DE AUTONOMIA: ",AU
110 IF AU>17 OR AU<1 THEN 100
120 CONTA(AU)=CONTA(AU)+1:A$(AU,CONTA(AU))=PR$:GOTO 70
200 INPUT "NRO. DE AUTONOMIA: ",NRO:IF NRO>17 THEN 200
210 FOR I=1 TO CONTA(NRO):PRINT A$(NRO,I);:NEXT I
```

(Ejercite el lector su imaginacion adelantandose al RUN.)

EDIT Nro. de línea de programa

De «Editar, corregir».

Comando no incluible en programa ni compilable.

Utilizable en correción de líneas de programa. (Véase explicación pormenorizada en capítulo 2 de la primera parte).

END

End = Fin.

Sentencia compilable.

No lleva aditamento ninguno, es sentencia monda y lironda.

Cuando el programa se encuentra con un END, termina irremisiblemente la ejecución de aquél. Si el programa está en intérprete, pasará a «modo control BASIC» y si está compilado, a «modo control sistema». Puede situarse en cualquier línea del programa, no necesariamente al final del mismo.

EOF (Nro. Fichero)

EOF son las iniciales de «End of File» = «Fin de Fichero».

Función numérica compilable.

Puede tomar sólo dos valores: 0 ó -1. Y esos valores no son asignables por el programador, los toma automáticamente cuando se trabaja con ficheros secuenciales. Su valor es siempre nulo a menos que se alcance el *final* del fichero, como consecuencia de una lectura del mismo, momento en el que queda cargado con -1 (ver última nota del comienzo de esta segunda parte).

EQV

De «Equivalent».

Operador lógico compilable, de muy escasa utilización.

Es el opuesto al XOR, es decir, cuando este operador va entre dos proposiciones lógicas, sólo si ambas son de la misma naturaleza (las dos verdaderas o las dos falsas) dará resultado «verdadero». Y si una de las dos (y sólo una) es falsa, el resultado será falso.

```
10 ' ****** EDIT ******
20 IMPUT "PRIMER APEELLIDO: ",AP1$
EDIT 20 <CR>
<S><M><C><N><1><4>< ><D><CR>
20 INPUT "PRIMER APELLIDO: ",AP1$
Ok

10 ' ****** END *******
20 R$=INPUT$(1):IF R$="F" THEN END
30 R=VAL(R$):ON R GOTO 100,200
40 END:'CUALQUIER VALOR DE R NO 1 ni 2
100 'RUTINA PARA R=1
- - - -
190 RETURN
200 'RUTINA PARA R=2
- - - -
315 RETURN


10 '******   EOF   ******
20 INPUT "Fichero a Ver (F=Fin): ",FICH$
25 IF FICH$="F" THEN END
30 OPEN "I", 1, FICH$
40 IF EOF(1) THEN CLOSE 1: GOTO 20
50 INPUT #1,A$: PRINT A$: GOTO 40

RUN <CR>
Fichero a Ver (F=Fin): SIEMPRE
SI LLORAS POR QUE HAS PERDIDO EL SOL
LAS LAGRIMAS
NO TE DEJARAN CONTEMPLAR LAS ESTRELLAS
Fichero a Ver (F=Fin): F
Ok


' *****  EQV  *****
PRINT  4>2 EQV 5<7:5>1 EQV 6<3:5<2 EQV 7>1
 -1   0   0
Ok
```

ERASE Lista de Variables Indexadas

ERASE = Borrar.

Sentencia *no* compilable.

Sentencia de las peligrosas pues deshace el efecto de la DIM en relación con las variables que acompañan a la ERASE. Con el uso de esta sentencia, las variables pueden volver a ser dimensionadas sin que aparezca el «Duplicate Definition». Sirve para liberar parte de la memoria interna.

No siendo compilable, creemos que no vale la pena extendernos más en esta sentencia.

ERL
ERR

«ERR» es abreviatura de «ERROR NUMBER» (número de error) y «ERL» de «ERROR LINE» (número de línea donde aparece el error).

Son variables del BASIC, reservadas.

Compilables, con reparos.

Se cargan automáticamente cuando se produce un error y existe una rutina de tratamiento de errores (ver ON ERROR). En ERR se carga el código del error ocurrido y en ERL el número de línea de programa donde ese error ha sido detectado.

Si el error se ha producido en una instrucción expresada en modo directo, la línea (ficticia, claro) donde se carga el error es la 65535.

El valor con que se ha cargado, tanto ERR como ERL, permanece en estas variables casi indeleblemente. Ni siquiera el CLR es capaz de borrarlas.

Dado el enorme consumo de memoria interna producido por un programa que sea compilado y que contenga rutinas de tratamiento de errores, recomendamos al lector-programador no haga uso de estas variables a menos que le sea absolutamente imprescindible.

```
10 ' ****** ERASE ******
20 DIM A(50),Z$(12)
30 FOR I=1 TO 50:A(I)=I*2/5:NEXT
32 FOR I=1 TO 10:PRINT A(I);:NEXT:PRINT
40 FOR I=1 TO 12:Z$(I)=STR$(A(I)):NEXT
50 ERASE A:DIM A(6)
60 FOR I=1 TO 6:PRINT A(I);:NEXT
70 DIM Z$(14)

RUN <CR>
 .4  .8  1.2  1.6  2  2.4  2.8  3.2  3.6  4
 0  0  0  0  0  0
Duplicate definition in 70

10 '****** ERL ERR ******
20 INPUT A: A$=A

RUN <CR>
?134
Type mismatch in 20

10 '****** ERL ERR ******
12 ON ERROR GOTO 5000
20 IF ERL=0 THEN INPUT A ELSE INPUT A$:END
22 A$=A
5000 '
5020 PRINT "ERROR NRO.";ERR;"DE INADECUADO USO DE VARIABLES "
5030 PRINT "EN LA IMPORTANTE LINEA";ERL:RESUME 20

RUN <CR>
?134
ERROR NRO. 13 DE INADECUADO USO DE VARIABLES
EN LA IMPORTANTE LINEA 22
?NADA
Ok
PRINT ERL;A$ <CR>
 22 NADA
Ok
NEW: PRINT A$;ERL <CR>
 22
```

ERROR Exp. Num.

Sentencia compilable.

Curiosa y de raro uso. Sirve para producir la simulación de errores cuyos códigos son definidos por el propio programador.

El valor de Exp. Num. debe estar comprendido entre 1 y 254.

Los códigos de error utilizados por el BASIC van, en las más recientes versiones, del 1 al 67, con algunas lagunas (algunos números libres).

Si el número que va junto a esta exótica sentencia no corresponde a ningún código de error existente, el mensaje que aparecerá es:

«Unprintable error» (Error sin mensaje)

El número (Exp. Num.) con el que hemos acompañado a ERROR queda cargado en la variable reservada ERR.

EXP (Exp. Num.)

Primeras letras de «Exponential de Exp. Num.».

Función numérica compilable.

Para programación estrictamente técnico-científica.

Esta función da el valor del antilogaritmo neperiano del argumento, es decir, el número *e* (base de los logaritmos neperianos) elevado a la potencia designada por Exp. Num.

Si este valor excede a la capacidad de la máquina, aparecerá un «Overflow» sin que se produzca interrupción del programa.

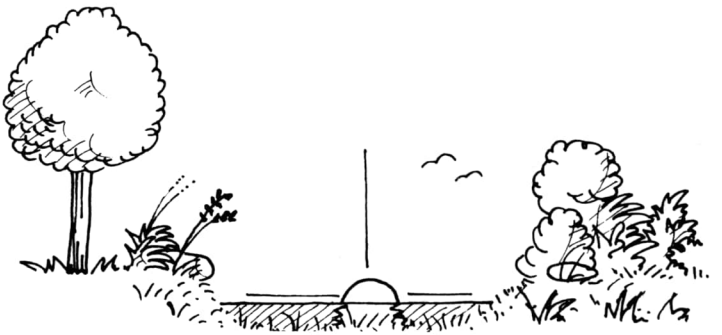
A menos que sea Vd. un científico, o esté estudiando bachillerato o sea profesor de matemáticas, le recomendamos se apresure a olvidar esta función ya mismo y sin contemplaciones. De nada.

```
5 ' ****** ERROR ******
10 INPUT "ERROR FORZADO QUE QUIERES QUE SALGA: ",E
20 ERROR E

RUN <CR>                                    ***
ERROR FORZADO QUE QUIERES QUE SALGA: 4      *
Out of DATA in 20
Ok
GOTO 10 <CR>                                ***
ERROR FORZADO QUE QUIERES QUE SALGA: 5      *
Illegal function call in 20
Ok
GOTO 10 <CR>                                ***
ERROR FORZADO QUE QUIERES QUE SALGA: 13     *
Type mismatch in 20
Ok
GOTO 10 <CR>                                ***
ERROR FORZADO QUE QUIERES QUE SALGA: 44     *
Unprintable error in 20
Ok


10 ' ******  EXP  ******
1000 INPUT X:IF X=999 THEN END
1010 PRINT EXP(X),EXP(X/7)
1020 PRINT EXP(1)
1040 GOTO 1000

RUN <CR>
?2
 7.38906      1.33071
 2.71828
?1.25
 3.49034      1.19551
 2.71828
?0
 1            1
 2.71828
?999
Ok
```

FIELD Fich. Num., longitud de campo AS Var. Alfanum.,...

Field = Campo.

Sentencia compilable de manejo de ficheros de acceso directo. Sirve para definir campos dentro de un registro cuya longitud total ha sido definida en el OPEN (apertura) del fichero. La longitud total o parcial de los campos definidos por el FIELD no debe en ningún caso sobrepasar la total del registro.



Las variables definidas por esta sentencia no pueden ser utilizadas en ningún otro tipo de asignaciones, sin incurrir en desastres impredecibles.

(Ver nota final de la introducción de esta segunda parte).

FILES Designación de disco Selección de ficheros

FILES = Ficheros.

Comando utilizable en programa y compilable.

Equivalente al comando de sistema (MS-DOS, CP/M, etc.) *DIR.*

Sin acompañamiento (FILES<CR>), da el listado de cuanto contiene el disco por defecto (el que estamos utilizando, con el que hemos entrado en el BASIC), ya sean programas del sistema, programas de utilidad, ficheros, programas Basic, id. Pascal, etc.

Para el caso de seleccionar los tipos de ficheros a visualizar, se tendrán en cuenta las siguientes normas:

- La designación completa (todo lo que vaya tras FILES) se escribirá entre comillas.
- Se podrá utilizar el asterisco (*) y el signo de interrogación a modo de comodines.

```
10 '****** FIELD ******
12 '
20 OPEN "R",2,"B:MANGURROS",64
22 '
30 FIELD 2, 32 AS AX$, 16 AS BX$, 16 AS CX$
40 FIELD 2, 64 AS X$
45 FIELD 2, 20 AS XX$
47 '
50 LSET X$="MANGULIBRIOS DE MANGO CORTO Y NO SOLO CORCHADO"
60 PUT 2,1
70 GET 2,1
80 PRINT AX$:PRINT BX$:PRINT CX$:PRINT XX$
90 CLOSE

RUN <CR>
MANGULIBRIOS DE MANGO CORTO Y NO
 SOLO CORCHADO

MANGULIBRIOS DE MANG
Ok


' ****** FILES ******

FILES <CR>
HAZPANT .COM  BRUN    .COM  L80     .COM  PP      .BAS
ENLACE  .COM  MANGURROS     PRUEBA        PROGR1  .BAS
Ok

FILES "B:*.BAS" <CR>
COSAS   .BAS  MAEXTRO .BAS  VENTAS  .BAS
Ok

FILES "A:*." <CR>
MANGURROS     PRUEBA
Ok

FILES "*.COM" <CR>
HAZPANT .COM  BRUN    .COM  L80     .COM  ENLACE  .COM
Ok
```

FIX (Exp. Num.)

Fixed Part of Exp. Num. (=Parte fija de Expresión numérica).

Función numérica compilable.

Muy semejante a la función INT, proporciona la parte entera de la Exp. Num. La ligera diferencia con el INT surge cuando se trata de aplicar estas funciones a valores negativos. En ellos, el valor que da el FIX es, estrictamente hablando, por exceso, mientras que el INT lo da por defecto. Con Exp. Num. positivas, ambos proporcionan las partes enteras por defecto.

La simple observación ocular de los ejemplos adjuntos esperamos que aclare definitivamente cualquier duda que el lector, muy justamente, pueda mantener.

FOR Var. Num. = Exp. Num. **TO** Exp. Num. [**STEP** Exp. Num.]

Traducción bastante libre:

«Para los valores de la variable numérica que aparece junto al FOR comprendidos entre la primera Exp. Num. y la segunda, y pasando de un valor a otro en saltos de magnitud igual a la Exp. Num. que acompaña a STEP (o de uno en uno si no existe STEP)».

Sentencia compleja compilable.

Generadora del bucle más clásico y frecuente del BASIC, esta sentencia lleva a cabo la ejecución de cuantas sentencias existan a continuación de ella (hasta topar con un NEXT Var. Num. que desempeña el papel de fin de ciclo). La ejecución se repite tantas veces como sean necesarias hasta que la Var. Num. alcanza (o sobrepasa) el límite prefijado en la segunda Exp. Num.

En el Basic de Microsoft, el control del programa puede salirse del ciclo por cumplimiento de alguna condición que, dentro de él, así lo requiera.

Dentro de un ciclo de esta naturaleza puede existir otro u otros ciclos anidados (uno dentro del otro) o adosados (uno junto a otro) siempre que no se produzcan intersecciones entre los distintos bucles.

(Véase para más abundamiento el capítulo 7 de la primera parte).

```
10 ' ******  FIX  ******
20 INPUT "NUMERO A REDONDEAR (0=FIN): ",NUM
25 IF NUM=0 THEN END
30 PRINT FIX(NUM);INT(NUM): GOTO 20

RUN <CR>
NUMERO A REDONDEAR (0=FIN): 17.84
 17  17
NUMERO A REDONDEAR (0=FIN): -34.71
-34 -35
NUMERO A REDONDEAR (0=FIN): -11.997
-11 -12
Ok


10 '******   FOR - NEXT   ******
20 'TABLA DE MULTIPLICAR DEL 7 SIN FOR-NEXT
30 A=A+1: N=7*A
50 PRINT "7 x";A;"=";N
60 IF A>9 THEN PRINT:GOTO 80
70 GOTO 30
80 ' Y ASI, LO MISMO PERO CON FOR-NEXT
90 FOR A=1 TO 10
100 PRINT "7 x";A;"=";7*A
110 NEXT A

5 '******   FOR-NEXT ANIDADO   ******
10 DIM A(8,3,12)
20 FOR I=1 TO 8:FOR J=1 TO 3:FOR K=1 TO 12
30   A(I,J,K)=I*J-K
40 NEXT K:NEXT J:NEXT I
50 FOR I=1 TO 8:FOR J=1 TO 3:FOR K=1 TO 12
60   PRINT A(I,J,K);
70 NEXT K:PRINT:NEXT J:NEXT I
RUN <CR>
 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11
 1  0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
 2  1  0 -1 -2 -3 -4 -5 -6 -7 -8 -9
 1  0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
- - - - - - - - - - - - - - - - - - - - - -
```

FRE (Argumento)

De FREE (libre, memoria libre en el sentido de disponible).

Función compilable cuyo Argumento es ficticio, es decir, puede Vd. poner entre paréntesis aquello que mejor le parezca. Si como argumento ponemos la cadena nula («») se produce un cierto reordenamiento de la memoria interna con alguna ganancia (a veces) de memoria disponible.

GET Fich. Num., Reg. Num.

GET (= conseguir, obtener. En nuestro caso equivale a *leer*)

Sentencia compilable. Se trata nada menos que de la sentencia mediante la cual, accedemos y leemos el contenido del registro número Reg. Num. del fichero abierto por el canal número Fich. Num.

Al ejecutarse esta sentencia, todas las variables definidas con la sentencia FIELD quedan automáticamente cargadas con los datos leídos en el fichero de disco correspondientes al citado número de registro. (Para más detalles nos remitimos a «FICHEROS BASIC»).

GOSUB Num. Lin.

GOSUB = GO to SUBroutine Num. Lin. (Vete a la rutina que da comienzo en la línea Num. Lin.)

Sentencia compilable de la mayor importancia.

Produce una bifurcación del control del programa (una separación del orden numérico estricto del programa) a la línea que le acompaña, con retorno automático a la instrucción siguiente a aquélla en la que estaba el GOSUB una vez alcanzada, dentro de la rutina, la sentencia RETURN. (Ver más en capítulo 4).

```
' ****** FRE ******

PRINT FRE(1);FRE("") <CR>
 31488  31488
Ok

LOAD "B:VENTAS": PRINT FRE("")
 8611
Ok

10 ' ****** GET ******
20 OPEN "R",3,"SOPUCIOS",20: FIELD 3, 20 AS SOPU$
40 FOR I=1 TO 10:LSET SOPU$=STRING$(20,I+64):PUT 3,I:NEXT
50 GET 3,1:PRINT SOPU$
60 GET 3:PRINT SOPU$
70 GET 3,10:PRINT SOPU$
80 CLOSE 3

RUN <CR>
AAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBB
JJJJJJJJJJJJJJJJJJJ
Ok

10 '****** GOSUB ******
20 INPUT "COSA A ANALIZAR: ",X$:IF X$="" THEN END
30 GOSUB 200:GOTO 20
200 'RUTINA BOBA 1
202 IF VAL(X$)<>0 THEN GOSUB 300:RETURN
210 PRINT "DATO NO NUMERICO":RETURN
300 'RUTINA BOBA 2
302 PRINT "DATO NUMERICO DE VALOR";VAL(X$):RETURN

RUN <CR>
COSA A ANALIZAR: 5528
DATO NUMERICO DE VALOR 5528
COSA A ANALIZAR: PEPE
DATO NO NUMERICO
COSA A ANALIZAR:
Ok
```

GOTO Num. Lin.

GOTO = GO TO Num. Lin. (Vete a la línea que te digo: Num. Lin.)

Sentencia compilable.

Cuando un programa se encuentra con un GOTO, continúa su ejecución en la primera instrucción que encuentre en la línea que acompaña al GOTO.

Si el GOTO no viene precedido por ninguna condición (ningún IF), se trata de una bifurcación *incondicional* a la línea indicada. Si, por el contrario, la instrucción contiene un IF anterior, estamos en presencia de una bifurcación *condicional.* En este caso, la bifurcación se producirá sólo si la condición especificada se cumple. En este tipo de bifurcaciones, es posible que se encuentre elíptico (en castellano, invisible) el GOTO. Son pues, equivalentes en el BASIC de Microsoft las tres expresiones;

```
IF A = 0 THEN GOTO 555
IF A = 0 GOTO 555
IF A = 0 THEN 555
```

HEX$ (Exp. Num.)

De «Hexadecimal».

Función alfanumérica (de argumento numérico) compilable.

Esta función (como OCT$) se limita a efectuar un simple cambio de base de numeración. En este caso convierte al número especificado en Exp. Num. en formato hexadecimal.

Sólo admite valores comprendidos entre 0 y 65535 (0 y FFFF en hexadecimal). No da error si el argumento es negativo pero el resultado de aplicar esta función a tal número es el complemento a 65536. Como en otros casos, le recomendamos que, a menos que se meta Vd. en programación muy especializada, olvide de inmediato esta función.

```
10 '******  GOTO  ******
20 A$=A$+"Z"
30 PRINT A$
40 GOTO 20

RUN <CR>
Z
ZZ
ZZZ
ZZZZ
ZZZZZ
- - - - -
String too long in 20

5 ' Muchos GOTOs para nada.
10 GOTO 100
50 GOTO 200
100 GOTO 50
200 G=G+1:IF G>2000 THEN END
210 GOTO 10

RUN <CR>
Ok

10 ' *****  HEX$  *****
20 INPUT "Numero (0=FIN): ",N:
30 IF N=0 THEN END
40 PRINT N;"DECIMAL=";HEX$(N);" HEXADECIMAL"
50 GOTO 20

RUN <CR>
Numero (0=Fin): 8
 8 DECIMAL=8 HEXADECIMAL
Numero (0=Fin): 255
 255 DECIMAL=FF HEXADECIMAL
Numero (0=Fin): 1024
 1024 DECIMAL=400 HEXADECIMAL
Numero (0=Fin): 0
Ok
```

IF

Su formato teórico completo es tan complicado como:

IF Expresión Relacional [y lógica] THEN Instrucción (es) [ELSE Instrucción (es)]

Se traduce por el también monosílabo *SI* condicional.

Sentencia compilable, que no funciona sola: habrá que adjuntarle alguna instrucción al final, para que cuando el IF, o sea la condición, se cumpla, ocurra algo perfectamente determinado.

Con esta sentencia, o mejor, con una instrucción que contenga un IF, el ordenador «decide» llevar a cabo una acción (la que se define tras el IF) u otra.

El cumplimiento de la condición puede dar lugar a una simple instrucción o a varias; incluso a otro IF. La instrucción o instrucciones que siguen al cumplimiento de la condición pueden contener bifurcaciones GOTO o GOSUB o cualquier otro tipo de instrucciones siempre que cumplan las reglas sintácticas del BASIC.

Por "Expresión Relacional" (la que sigue al IF) entendemos cualquier comparación en la que intervenga cualquiera de los seis operadores relacionales conocidos:

$$= \quad > \quad < \quad > = \quad < = \quad <>$$

El IF hace una evaluación de la expresión de la que, sin posible ambiguedad, sale:

el valor **-1** si la condición se cumple
el valor **0** si la condición no se cumple

Cuando la expresión es del tipo X<>0 y X es un "entero", se puede uno ahorrar el<>0, es decir son equivalentes las instrucciónes IF X<>0 THEN... y IF X THEN...

La expresión relacional puede ir enlazada con otra u otras mediante operadores lógicos (AND, OR, etc.).

Recomendamos al lector se desplace al capítulo 4 de la primera parte si se ha quedado un poco a dos velas aquí.

```
10 ' ****** IF ******
20 INPUT "Dime un ano (CR Fin): ",AN$:IF AN$="" THEN 80
30 A=VAL(AN$):IF A=0 THEN 20 ELSE T$="NO"
40 IF A MOD 4=0 AND A MOD 100 OR A MOD 1000=0 THEN T$=""
50 PRINT A;T$;" ES BISIESTO":GOTO 20
80 DATA 10,5,8,12,25,33,2,1
90 DEFINT A:DIM A(8):PRINT
100 FOR I=1 TO 8:READ A(I):NEXT I:GOSUB 190
110 'Ordenemos de menor a mayor estos numeros.
120 FOR I=1 TO 7
130     FOR J=I+1 TO 8
140             IF A(J)<A(I) THEN SWAP A(J),A(I)
150     NEXT J
160 NEXT I: GOSUB 190:GOTO 210
180 ' Rutinilla
190 FOR I=1 TO 8:PRINT A(I);:NEXT I:PRINT:RETURN
210 'Numeros capicuas
220 INPUT "Numero (CR=Fin): ",N$:IF N$="" THEN END
230 'Damos la vuelta al numero.
240 A$="":FOR I=LEN(N$) TO 1 STEP -1
250 X$=MID$(N$,I):A$=A$+LEFT$(X$,1): NEXT I
270 T$="":IF A$<>N$ THEN T$="NO "
280 PRINT N$;" ";T$;"ES CAPICUA":GOTO 220

RUN <CR>
Dime un ano (CR=Fin): 1986
 1986 NO ES BISIESTO
Dime un ano (CR=Fin): 2000
 2000 ES BISIESTO
Dime un ano (CR=Fin): 1988
 1988 ES BISIESTO
Dime un ano (CR=Fin):
 10  5  8  12  25  33  2  1
 1  2  5  8  10  12  25  33
Numero (CR=Fin): 12121212
12121212 NO ES CAPICUA
Numero (CR=Fin): 1212121
1212121 ES CAPICUA
Numero (CR=Fin):
Ok
```

IMP

De "Implication". Operador lógico compilable de rarísimo uso.

Situado este operador entre dos proposiciones lógicas el resultado es siempre verdadero a menos que la segunda **y sólo la segunda** proposición sea falsa.

INCLUDE (Véase Compilación, Anexo 2)

INKEY$

Traducible por "En la tecla". (Lo que nos deja como estabamos).

Función alfanumérica compilable. No lleva argumento. Semejante (y uso un poco más engorroso) que la función INPUT$(1) (recomendamos que se emplee éste en lo posible), su utilización está prevista para interrupciones voluntarias y provisionales del programa.

Si se carga una variable alfanumérica con esta función, la variable queda automáticamente anonadada (convertida en ""). Es necesario, para que esta función cumpla con su objetivo interruptorio, provocar un bucle cerrado sobre sí mismo.

INP (Exp. Num.)

Primeras letras de INPUT, con la que no debe confundirse.

Función numérica compilable. La Exp. Num. debe contener un número de "puerta" de entrada/salida del ordenador, número reconocible por el sistema y comprendido entre 0 y 65535.

Esta función está destinada a utilizarse en programas en que vaya a producirse una interacción con el "exterior" en un sentido muy amplio. con esta función se "lee" algo que procede de ese exterior. Se trata por ello de un escarceo del BASIC en el mundo de la robótica o, al menos de las comunicaciones.

En INP (Exp. Num.) queda incorporado el valor del byte que la función encuentra en la puerta Exp. Num. (Ver OUT).

El parentesco de INP con INPUT es obvia. Digamos que con el INPUT, el "exterior" (poco, desde luego) es el teclado, desde donde se introduce el dato correspondiente.

```
'*******   IMP   *******
PRINT 4<3 IMP 3>4:4>3 IMP 3>4:4<3 IMP 3<4:4>3 IMP 3<4
-1  0 -1 -1

10 '******   INKEY$   ******
15 A=5
20 FOR S=A TO 10000 STEP A: PRINT S;S/4
40 IF INKEY$<>"" THEN 70
50 NEXT S
60 '
70 Y$=INKEY$:IF Y$="F" THEN END ELSE IF Y$="" THEN 70
80 A=A+1:GOTO 20

RUN <CR>
 5  1.25
 10  2.5
 15 3.75   <-------- Hemos pulsado una tecla y luego otra <>F
 6  1.5
 12  3     <-------- Hemos pulsado una tecla y luego una F.
 Ok

10 ' ******  INP  ******
20 'Rutina de lectura de un dato que llega
30 'de un dispositivo exterior (por ejemplo un MODEM)
40 'conectado al ordenador a traves de una puerta serie.
50 '
60 CONTROL=65
70 'La puerta de control de la entrada es la 65
80 DATOS=64
90 'La puerta de datos de entrada es la 64
100 STATUS=INP(CONTROL)
110 IF STATUS AND 1 = 0 THEN 100
120 DATO=INP(DATOS)
130 PRINT DATO;
140 'En la variable DATO queda cargado el valor
150 'encontrado en la puerta de datos.

RUN <CR>
 116
Ok
```

INPUT [;] [Mensaje alfanum.,] Variable [,Variable, Variable...]

INPUT = Entrada, imputación

Sentencia compilable.

Permite introducción de datos durante la ejecución del programa quedando cargada(s) la(s) variable(s) que acompaña(n) al INPUT.

Cuando, en el curso de ejecución de un programa, éste se tropieza con un INPUT, el programa se interrumpe a la espera de que el operador teclee algo. Justo antes de la primera variable, puede haber una coma (,) o un punto y coma (;). Si hay una coma, la petición de dato aparecerá sin signo de interrogación, signo que, en cambio aparecerá si hay un punto y coma.

Justo detrás de la sentencia INPUT puede ponerse también un punto y coma. Con él, después de pulsado CR (fin de introducción), el cursor permanece en la misma línea, mientras que si ese signo no se pone, se produce, tras el CR, un salto a la línea siguiente.

El mensaje opcional que puede escribirse junto al INPUT consistirá en un mensaje-constante, es decir, debe ser una cadena alfanumérica concreta (la que sea).

Cuando se pretenda introducir más de una variable con el mismo INPUT, tanto en el programa como en la introducción de datos deben situarse las correspondientes comas.

Los datos que el operador introduzca merced a esta sentecia no deben entrecomillarse.

Los datos tecleados han de corresponder en naturaleza a los de las variables del programa. Concretamente, no podremos introducir «PEPE» si la variable INPUT es X, ni tampoco 100000 si la variable es X%; si así lo hacemos se presentará en pantalla un "Redo from start?" que, sin abortarnos el programa, dejará muy mal parada a la pantalla.

(El inconveniente anterior, junto con otros cuyo detalle quedan analizados en el capítulo 8 de la primera parte, constituyen el motivo por el que debe limitarse al máximo el uso de esta sentecia sustituyendose por una cadena de INPUT$(1) tal como en dicho capítulo se explica).

```
10 '****** INPUT ******
20 PRINT "CON INTERROGACION Y SIN CARTEL (0 termina)"
30 INPUT X:IF X=0 THEN 60 ELSE PRINT:GOTO 20
60 PRINT
70 PRINT "SIN INTERROGACION Y SIN CARTEL"
80 INPUT "",X
85 PRINT
90 INPUT "SIN INTERROGACION (Y CON ESTE CARTEL): ",X$
100 PRINT
110 INPUT "CON INTERROGACION (Y CON CARTEL): ";Y$
120 PRINT
130 '
140 'INPUT MULTIPLE
150 INPUT "Nombre, Direccion, Codigo Postal: ",N$,D$,COD$
160 PRINT LEN(N$);LEN(D$);LEN(COD$)
170 INPUT "Seguimos ";S$
175 G$=LEFT$(S$,1)
180 IF G$="S" OR G$="s" THEN 150

RUN <CR>
CON INTERROGACION Y SIN CARTEL (0 termina)
?234

CON INTERROGACION Y SIN CARTEL (0 termina)
?0

SIN INTERROGACION Y SIN CARTEL
234

SIN INTERROGACION (Y CON ESTE CARTEL): 234

CON INTERROGACION (Y CON CARTEL): ?234

Nombre, Direccion, Codigo Postal: CARLOS, PEZ 10, 28007
 6  6  5
Seguimos ?s
Nombre, Direccion, Codigo Postal:          PEPE,A,28007
 4  1  5
Seguimos ?N
Ok
```

INPUT$ (Exp. Num. [,Exp. Num.])

En una forma más reducida y práctica podríamos poner:

INPUT$(N) en la que N es un número entero mayor de 0 y menor de 256.

Función alfanumérica de argumento numérico. Compilable.

Como la sentencia INPUT, esta función provoca una interrupción del programa hasta que se pulsen tantas teclas como indique el argumento.

Si el argumento es igual a 1, el programa proseguirá tan pronto se toque una tecla cualquiera que sea activa, es decir, no vale pulsar la tecla de mayúsculas o la de control en solitario. Sí, en cambio se podrá pulsar CONTROL-A o CONTROL-J.

Si se pulsa CONTROL-C, estando en intérprete, quedará abortado el programa.

Cualquiera que sea el valor del argumento, las teclas pulsadas no tendrán su reflejo en pantalla (no tendrán "eco").

También se puede utilizar, en lugar del teclado, el disco o diskette como entrada de los caracteres requeridos, a través de una lectura directa —mediante esta función— de los registros de un fichero secuencial. En ese caso, pondríamos, por ejemplo:

```
A$=INPUT$ (1,3)
```

que significa: métase en A$ el primer carácter (el siguiente, si estamos en un ciclo) de lo leido en el fichero abierto por el canal 3.

El caso particular INPUT$(1) tiene dos aplicaciones de excepcional importancia:

- Sustituye con ventaja a la función INKEY$.
- Es base de trabajo para una buena rutina de INPUT.

(Repásense los capitulos 6 y 8 en lo que concierne a esta crucial función y estúdiense con detenimiento los ejemplos que figuran a la derecha).

```
20 ' ****** INPUT$(X) ******
30 DIM A$(12)
40 DATA ENERO,FEBRERO,MARZO,ABRIL,MAYO,JUNIO,JULIO
60 DATA AGOSTO,SEPTIEMBRE,OCTUBRE,NOVIEMBRE,DICIEMBRE
82 FOR I=1 TO 12:READ A$(I):NEXT I
84 '
90 PRINT "HACIA ADELANTE [+]   HACIA ATRAS [-]    FIN [0]"
100 Y$=INPUT$(1)
102 IF Y$="0" THEN 310
110 IF Y$="+" THEN N=N+1:IF N>12 THEN N=1
120 IF Y$="-" THEN N=N-1:IF N<1 THEN N=12
122 IF Y$<>"+" AND Y$<>"-" THEN 100
160 PRINT A$(N): GOTO 100
310 PRINT "GRABAR [G]   LEER [L]   MODIFICAR [M]   FIN [F]"
320 I$="GgLlMmFf":Y$=INPUT$(1)
330 I=INSTR(I$,Y$):I=(I+1)\2:ON I GOTO 400,500,600,1000
340 GOTO 310
400 PRINT "PROGRAMA DE GRABACION"
490 Y$=INPUT$(1):GOTO 310
500 PRINT "PROGRAMA DE LECTURA":GOTO 490
600 PRINT "PROGRAMA DE MODIFICACION":GOTO 490
1000 END

RUN <CR>
HACIA ADELANTE [+]   HACIA ATRAS [-]    FIN [0]
ENERO
FEBRERO
MARZO
FEBRERO
ENERO
DICIEMBRE
NOVIEMBRE
GRABAR [G]   LEER [L]   MODIFICAR [M]   FIN [F]
PROGRAMA DE MODIFICACION
GRABAR [G]   LEER [L]   MODIFICAR [M]   FIN [F]
PROGRAMA DE GRABACION
GRABAR [G]   LEER [L]   MODIFICAR [M]   FIN [F]
PROGRAMA DE LECTURA
GRABAR [G]   LEER [L]   MODIFICAR [M]   FIN [F]
Ok
```

INPUT # Num. Fich., Variable[,Variable,...]

"Entrada por el Canal expresado en Num. Fich."

Sentencia compilable.

Con esta sentencia se lee, no lo que se da por teclado (eso lo hacía esa misma sentencia pero sin el signo " ; "), sino lo que está grabado en un fichero secuencial abierto por el canal Num. Fich. (abierto con el ese número de fichero).

El tipo de variable leida debe coincidir con el que acompaña a la sentencia.

(Esta sentencia, como todas las que corresponden a ficheros están ampliamente desarrolladas en "FICHEROS BASIC" y "APLICACIONES DE FICHEROS BASIC", de esta misma colección. Nos limitamos aquí a proporcionar una síntesis como herramienta de consulta rápida para el lector-programador).

INSTR ([Exp. Num.,]Exp. Alf.,Exp.alf.)

Contracción de "INternal Occurrence of STRing" que podemos traducir muy libremente por "Existencia y posición de una subcadena dentro de una cadena".

Función numérica de argumento alfanumérico, compilable.

En esta función queda cargado el número de posición (número de orden) que ocupa dentro de una cadena, otra cadena alfanumérica cualquiera. El número que proporciona el INSTR es el que corresponde al primero de la subcadena.

El valor numérico de la Exp. Num. indica el número de orden de los caracteres que componen la cadena principal, **a partir del cual** dá comienzo la búsqueda de la subcadena. Si no se pone el primer parámetro (Exp. Num.), el valor por defecto es la unidad. Esto significa que la búsqueda se incia en el mismo comienzo de la cadena principal.

Si la subcadena no existe dentro de la cadena, el valor del INSTR es cero. Es de resaltar la fulminante velocidad de ejecución de esta función. (Compruébese en los ejemplos).

```
10 ' ******   INPUT #   ******
20 OPEN "I",1,"BECQUER"
30 FOR I=1 TO 4:INPUT #1, A$:PRINT A$:NEXT
40 CLOSE

RUN <CR>
VOLVERAN LAS OSCURAS GOLONDRINAS
DE TU BALCON SUS NIDOS A COLGAR
Y OTRA VEZ CON EL ALA EN LOS CRISTALES
JUGANDO LLAMARAN.
Ok


5  ' ******   INSTR   ******
10 A$="fffffffffffffffffffffffffffffffffffffffffffffffffffffa"
12 PRINT CHR$(7)
20 FOR I=1 TO 100: B=INSTR(A$,"a"): NEXT
50 PRINT CHR$(7);B

RUN <CR>
 50

10 DIM POSI(100):WIDTH 255
20 INPUT "ESCRIBE UNA FRASE (F=Fin): ",FRASE$
30 IF FRASE$="F" THEN END
40 PRINT "ESCRIBE LO QUE QUIERES BUSCAR DENTRO DE ";FRASE$;
50 INPUT ": ",X$
60 N=1:VECES=0
70 A=INSTR(N,FRASE$,X$):IF A=0 THEN 100 ELSE VECES=VECES+1
80 POSI(VECES)=A:N=A+1
90 GOTO 70
100 VEZ$="VECES":IF VECES=0 THEN VEZ$="(NINGUNA VEZ)"
110 IF VECES=1 THEN VEZ$="VEZ."
120 PRINT:PRINT "EN ";FRASE$;", ";
125 PRINT X$;" ESTA CONTENIDA";VECES;VEZ$:IF VECES=0 THEN 140
130 PRINT "EN LAS POSICIONES ";:FOR I=1 TO VECES:PRINT POSI(I);:NEXT
140 PRINT:PRINT:GOTO 20
```

(Ponga a prueba el lector su fertil imaginacion.)

INT (Exp. Num.)

"Integer part of Exp. Num." = "Parte entera de Exp. Num."

Función numérica compilable.

Da la parte entera del argumento, es decir de la expresión numérica Exp. Num.

La parte entera de un número positivo es algo obvio. La de un número negativo ya es más discutible. Podemos decir que es como si nos pusieramos en plan pesimista. Si "tenemos" 427 y pico, su parte entera es 427 y si "debemos" (negativo) 427 y pico, su parte entera es -428.

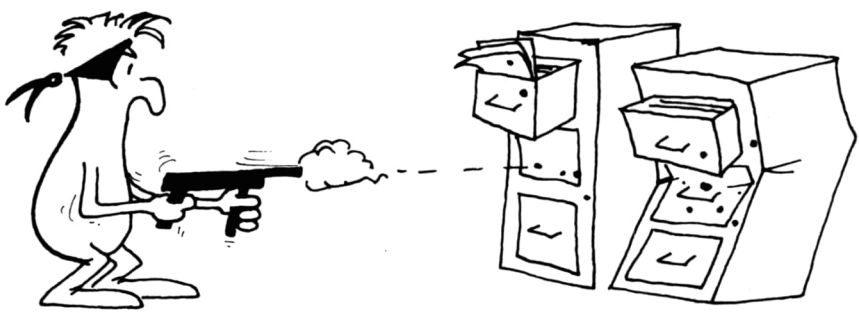
Muy semejante al FIX. Véase éste y los ejemplos adjuntos.

KILL (Nombre de Fichero)

Del dramático verbo kill (=matar, asesinar).

Comando incluible en programa y compilable.

Elimina del directorio el fichero o programa especificado con el KILL. Este nombre de fichero debe escribirse entre comillas.



El BASIC de CP/M distingue, en el directorio del disco, las mayúsculas y las minúsculas, cosa que no hace el sistema operativo. (O sea, que se puede salvar un programa como "PEPE", otro como «pepe», otro distintivo como «Pepe», etc.). Esto no es posible cuando se trata del Basic de MSDOS (16 bits) donde no hay distinción entre mayúsculas y minúsculas: todo es traducido a mayúsculas. Consecuencia práctica: con el Basic de CP/M, un programa (o, en general un fichero) salvado (grabado) desde el intérprete que contenga minúsculas, solo puede ser destruido también desde el BASIC.

```
10 ' ******  INT  ******
12 '
20 INPUT "CIFRA CON DECIMALES (0=FIN):",N
30 IF N=0 THEN END
40 PRINT "PARTE ENTERA DE ";N;"= ";INT(N)
50 GOTO 20

RUN <CR>
CIFRA EN DECIMALES (0=FIN): 7.71
PARTE ENTERA DE  7.71 =  7
CIFRA EN DECIMALES (0=FIN): -13.0002
PARTE ENTERA DE -13.0002 = -14
CIFRA EN DECIMALES (0=FIN):999.999
PARTE ENTERA DE  999.999 =  999
CIFRA EN DECIMALES (0=FIN): 0
Ok

KILL "B:PEPORRO" <CR>
Ok
KILL "PROGRAM.BAS" <CR>
Ok

10 PRINT "ANTES DE MATAR A PEPE:"
13 FILES "A:*."
20 KILL "PEPE"
22 PRINT
25 PRINT "DESPUES DE MATAR A PEPE:"
30 FILES "A:*."

RUN <CR>
ANTES DE MATAR A PEPE:
CASPITA        MANOLEZ        PEPE        RAM

DESPUES DE MATAR A PEPE:
CASPITA        MANOLEZ        RAM
Ok

KILL "PEPE"
File not found
Ok
```

LEFT$ (Exp. Alf., Exp. Num.)

"LEFT part of Exp. Alf." = "Parte izquierda de Exp. Alf."
Función alfanumérica compilable.

Esta función, aplicada a una cadena cualquiera Exp. Alf., toma tantos caracteres por la izquierda de esa cadena como indique el valor numérico de Exp. Num.

Puede ocurrir:

— Que el contenido de Exp. Alf. sea nulo ("cadena nula"). En ese caso, ya puede valer valer Exp. Num. lo que a Vd. se le ocurra. La función contendrá siempre también la cadena nula ("").

— Que la longitud de la cadena (Exp. Alf.), es decir su número de caracteres (vease LEN), sea inferior al valor de Exp. Num. En tal caso la función toma todos los caracteres de Exp. Alf. sin producir error.

— Que el valor de Exp. Num. sea cero. La función LEFT$ en este caso pasa a convertirse en la cadena nula ("").

— Por último, que el valor de Exp. Num. sea negativo. Sólo en ese caso se produce error:

Illegal function call in 335

La función LEFT$ junto con RIGHT$ y MID$, constituyen el trio de funciones más típicos y útiles para el manejo adecuado de las cadenas alfanuméricas. De ellos, la función MID$ es la más completa de las tres. En concreto cualquier expresión del tipo LEFT$(X$,X) es idéntica a MID$(X$,1,X).

(Para mayor información, acúdase al capítulo 5 de la primera parte donde se desmenuzan todos estos conceptos).

```
10 ' ****** LEFT$ ******
20 INPUT "ANDA, ESCRIBE ALGO: ",PP$
30 IF PP$="" THEN END
40 PRINT LEFT$(PP$,4);" "; MID$(PP$,1,4)
60 GOTO 20

RUN <CR>
ANDA, ESCRIBE ALGO: RAMON ZABALA
RAMO RAMO
ANDA, ESCRIBE ALGO: PATOLOGIA CIMBORRICA
PATO PATO
ANDA, EXCRIBE ALGO:
Ok

10 'UN INTENTO DE PRINT USING CASTELLANO
20 DEFDBL N
30 INPUT "NUMERO (0=FIN): ",NUM: IF NUM=0 THEN END
40 N=ABS(NUM):N$=STR$(N)
50 IF N=INT(N) THEN N1$=N$:N2$=",-":GOTO 70
60 D=INSTR(N$,"."):N1$=LEFT$(N$,D-1):N2$=","+MID$(N$,D+1)
70 IF N<1 THEN N1$="0"
80 IF N<1000 THEN 100
90 N1$=LEFT$(N1$,LEN(N1$)-3)+"."+RIGHT$(N1$,3)
100 RELL=40-LEN(N1$):N$=SPACE$(RELL)+N1$+N2$
110 IF NUM>=0 THEN 140 ELSE IF N>=1 THEN RELL=RELL+1
130 MID$(N$,RELL)="-"
140 PRINT N$:GOTO 30

RUN <CR>
NUMERO (0=FIN): 4321
                                    4.321,-
NUMERO (0=FIN): -12232.37
                                  -12.232,37
NUMERO (0=FIN): 1.33
                                       1,33
NUMERO (0=FIN): 973765
                                  973.765,-
NUMERO (0=FIN): 0
Ok
```

LEN (Exp. Alf.)

De Length (longitud)

Función numérica de argumento alfanumérico. Compilable.

Proporciona el número de caracteres que contiene la Exp. Alf. sin más. Si el argumento es la cadena nula, el valor de su LEN es cero.

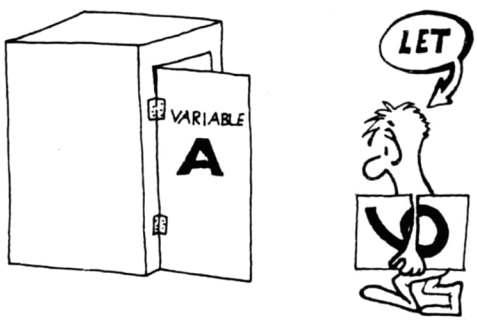
El valor de esta función en casi todos los BASICs tiene como límites 0 y 255 (máxima longitud de un "string"). En recientes versiones de MSDOS (16 bits), el valor que puede tomar esta función (teórico, pues es seguro que no llegará a tanto la capacidad de la memoria interna) es de 65535 bytes.

Si el argumento contiene una expresión numérica, caeremos en un Illegal function call que abortará rudamente nuestro bienamado programa.

[LET] Variable = Expresión

Let = Sea.

Sentencia compilable. Se trata de una sentencia que puede hacerse elíptica, es decir, puede omitirse. De hecho, no se utiliza nunca, pues buena gana de trabajar en balde. Sirve para asignar valores a variables y vectores de todo tipo.



Procede de las primeras versiones de BASIC y se mantiene por una cuestión de compatibilidad con anteriores intérpretes. Apresúrese el lector a condenarla al olvido.

(En el capítulo 3 de la primera parte se dan las normas de asignación de variables de todo tipo, incluidas las indexadas).

```
' ****** LEN ******
A$="KAKARRACO":PRINT LEN(A$);:A$="":PRINT LEN(A$)
 9  0
Ok

10 'CENTRADO DE ROTULOS
20 INPUT "ROTULO: ",RO$:IF RO$="" THEN END
30 PRINT TAB(40-(LEN(RO$)/2));RO$:GOTO 20

RUN <CR>
ROTULO: MARIA DOLORES
                                MARIA DOLORES
ROTULO: CLIENTE
                                   CLIENTE
ROTULO: PAF
                                     PAF
ROTULO:
Ok

10 ' ****** LET ******
20 LET A=5:PRINT A;
40 A=7:PRINT A;
60 B=5:LET C=2:LET A=1
70 LET D=SQR(B^2-4*A*C): PRINT D
82 '
85 'SUMA DE LOS N PRIMEROS NUMEROS NATURALES
90 INPUT "HASTA NUMERO: ",N:IF N=0 THEN END
100 LET SUMA=0: LET X=0
110 LET X=X+1:IF X>N THEN PRINT:GOTO 90
120 LET SUMA=SUMA+X:PRINT SUMA;:GOTO 110
140 '(Si quitamos todos los LET, todo seria igualito.)

RUN <CR>
 5  7  4.12311
HASTA NUMERO: 5
 1  3  6  10  15
HASTA NUMERO: 12
 1  3  6  10  15  21  28  36  45  55  66  78
HASTA NUMERO: 0
Ok
```

LINE INPUT[;] [Mensaje;] Variable Alfanumérica

"Entrada de línea"

Sentencia compilable.

Si se la compara con INPUT (hágase, si no es gran molestia), vemos que se le parece bastante.

Como el INPUT, permite la introducción de una "línea" completa (hasta 254 caracteres) en una sola variable. Las diferencias con el INPUT ordinario son éstas:

- Con el INPUT pueden acompañarse diversas variables separadas por comas. El LINE INPUT en cambio sólo admite una variable, precisamente alfanumérica (o elemento de vector alfanumérico).
- Los espacios en cabeza y cola del INPUT son fulminantemente despreciados por éste. Con el LINE INPUT, estos espacios "cuentan", es decir, entran en la variable del mismo.
- Los separadores (coma, punto y coma) no son admisibles por el INPUT como parte integrante de la introducción, cosa que si sucede con el LINE INPUT.

Aparte de estas diferencias, el funcionamiento de esta sentencia respecto al resto del formato, es idéntico al del INPUT. (Ver capítulo 8 de la primera parte donde puede el lector completar información respecto al LINE INPUT).

LINE INPUT # Nro. Fich., Var. Alfan.

"Entrada de línea procedente del fichero secuencial Nro. Fich.".

Sentencia compilable.

Muy semejante a la anterior, en la variable alfanumérica se carga, en lugar de aquello que se escribe en el teclado, el contenido del registro *siguiente* del fichero secuencial abierto previamente con un OPEN "I" (para lectura) por el canal (con el número) Nro. Fich. (Ver nota inicial relacionada con Tratamiento de Ficheros).

```
20 '****** LINE INPUT ******
40 INPUT "COMO INPUT: ",A$: PRINT A$;LEN(A$)
80 LINE INPUT "COMO LINE INPUT: ",B$: PRINT B$;LEN(B$)
120 INPUT "INPUT CON COMAS: ",A$,B$
140 PRINT A$;" ";B$
160 LINE INPUT "LO MISMO CON LINE INPUT: ",A$
180 PRINT A$
200 'INSTRUCCIONES IMPOSIBLES: LAS DOS QUE SIGUEN
220 LINE INPUT A$,B$
240 LINE INPUT X

RUN <CR>
COMO INPUT:    CASPE
CASPE 5
COMO LINE INPUT:    CASPE
   CASPE 8
INPUT CON COMAS: CALATRAVA,   MONTESA
CALATRAVA MONTESA
LO MISMO CON LINE INPUT: CALATRAVA,   MONTESA
CALATRAVA,   MONTESA
Syntax error in 220

RUN 240 <CR>
Illegal function call in 240
Ok

10 '****** LINE INPUT # ******
20 OPEN "I", 2, "B:COSAS"
40 IF EOF(2) THEN END
60 LINE INPUT #2, A$: PRINT A$: GOTO 40

RUN <CR>
Me faltas tu, mujer,
y por eso la vida se me pierde en los vientos que callan
Te llamo por las noches
cuando se me desborda la tristeza en el alma
Y sonando contigo
escucho silenciosas palabras que me hablas.
(E.DURAN)
Ok
```

LIST [Num. Lin.] [-Num. Lin.]

De «Listar».

Comando incorporable en programa y no compilable.

Saca por pantalla el listado del programa que esté presente en memoria interna. Si se escribe LIST a secas, saldrá el listado completo —a menos que lo interrumpamos provisional (CONTROL-S) o definitivamente (CONTROL-C).

Si junto al LIST tecleamos un número de línea, ésta hará su aparición en solitario (si es que existe). Si además, escribimos un guión, la salida que se produce es el de todas las líneas de programa que haya presentes a partir del número especificado hasta el final.

Si escribimos junto al LIST un guión y *después* un número de línea, se visualizará el listado del programa desde el comienzo del mismo hasta la línea deseada inclusive.

Si, por último, tecleamos el comando acompañado de dos líneas (la primera menor que la segunda), separadas por un guión, aparecerá el listado del programa comprendido entre esas dos líneas.

Si el comando forma parte de un programa, las instrucciones que pueda haber a continuación quedan inoperantes ya que la aparición de un LIST dentro de una línea, después de efectuar un listado del programa en cuestión, pone al sistema en "modo control BASIC".

LLIST [Num. Lin.] [-Num. Lin.]

De «Line-printer-terminal LIST» = Listar por Impresora.

Comando incorporable en programa (no se nos alcanza para qué) y no compilable, claro.

Se trata del mismo comando anterior sólo que la salida o salidas del listado del programa no se produce en la pantalla sino en la impresora que, naturalmente, debe estar instalada («colgada» para que pareza más informático) y conectada («encendida»).

```
2 '****** LIST ******
4 '
6  PRINT "HOY LOS CIELOS Y LA TIERRA ESTAN CUBIERTOS"
8  PRINT "HOY HA LLEGADO AL COLMO MI GRADO DE IDIOTEZ"
10 PRINT "HOY LA HE VISTO, LA HE VISTO E IBA CON OTRO"
12 PRINT "ME CACHIS DIEZ!"
14 '
16 PRINT "MUJERES..."
18 PRINT "MUJERES, PERFIDOS SERES"
20 PRINT "QUE ENVENENASTEIS MI VIDA"
22 PRINT "MENOS TU, QUE TU NO ERES"
24 PRINT "COMO LAS DEMAS MUJERES"
26 PRINT "ERES PEOR TODAVIA !"
28 END:'Y VDS. DISPENSEN

LIST 26- <CR>
26 PRINT "ERES PEOR TODAVIA !"
28 END:'Y VDS. DISPENSEN
Ok

LIST -9
2 '****** LIST ******
4 '
6  PRINT "HOY LOS CIELOS Y LA TIERRA ESTAN CUBIERTOS"
8  PRINT "HOY HA LLEGADO AL COLMO MI GRADO DE IDIOTEZ"
Ok


10 '***** LLIST *****
20 'Una vez que este valioso programa
30 'haya sido debidamente confeccionado
40 'le someteremos a un "LLIST"
50 'a ver que pasa.
60 A=3:B=A:C=INT(314/14)
70 END

LLIST <CR>
(Si tenemos conectada la impresora, veremos salir por ese
periferico, con su cantarin acento de grajo, el listado
del muy ingenioso programa anterior.)
```

LOAD Nombre de Programa[,R]

De Load = Cargar.

Comando incluible en programa y compilable con reservas (ver).

Sin la opción final [,R], este comando se ocupa de buscar en el disco especificado en el comienzo de «Nombre de Programa» (o en el disco/diskette por defecto si nada se especifica), el programa en cuestión y trasladarlo a la memoria interna. Esta introducción del programa tiene lugar con un previo borre de todo el area de usuario de esa memoria (excepto variables de error del BASIC).

Con la opción ",R" se produce el mismísimo efecto que si escribiéramos "RUN", es decir, el programa, además de pasar de memoria externa a memoria interna se lanza a su ejecución.

Las "reservas" en cuanto a su compilación consisten simplemente en que podemos incluir este comando en un programa que va a ser compilado, sólo con la opción ",R".

LOC (Nro. Fich.)

De "Location" = Situación.

Función numérica compilable.

Se utiliza raras veces. Función específica de tratamiento de ficheros. Dependiendo del tipo de fichero a que se hace referencia, esta función da:

— Si el fichero es de acceso directo, el número del último registro *leido o grabado* (ver GET y PUT).

— Si se trata de un fichero secuencial, también el último registro leido o grabado pero esta vez *registro físico, de 128 bytes.*

(Ver nota inicial respecto a Ficheros).

```
' ******  LOAD  ******
NEW <CR>
Ok
LIST <CR>
Ok
LOAD "PEPON" <CR>
Ok
LIST <CR>
20 PRINT "PEPON ";
40 IF N>=4 THEN PRINT:PRINT "YA VALE":END
50 N=N+1:GOTO 20
Ok
NEW: LIST <CR>
Ok
LOAD "PEPON",R <CR>
PEPON PEPON PEPON PEPON PEPON
YA VALE
Ok

20 '******  LOC  ******
40 OPEN "R",1,"B:MANGURROS",10
50 N=0:FIELD 1,10 AS X$
100 WHILE A$<>SPACE$(10): GET 1:N=N+1:A$=X
200 PRINT X$: WEND: PRINT LOC(1):N
280 OPEN "I",2,"B:COSAS"
300 IF EOF(2) THEN CLOSE:END
320 LINE INPUT#2,A$:PRINT A$;LOC(2):GOTO 3
RUN <CR>
AAAAAAAAAA
AABBBBBBBB
BBBBCCCCCC
CCCCCC
 4  4
Anoche, cuando dormia 1
sone, bendita ilusion 1
que una colmena tenia 1
dentro de mi corazon. 1
Y las doradas abejas, 1
iban fabricando en el 2
Ok
```

LOG (Exp. Num.)

De Logaritmo natural (neperiano).

Función numérica apta para ser compilada.

Aplicada esta función a un número o expresión numérica nos devuelve el valor del logaritmo neperiano de ese número. Si el número es 0 o negativo, saltará error ("Illegal function call").

LPOS (z)

De "Posición en Impresora".

Función compilable de uso poco frecuente.

El argumento es ficticio, podemos poner lo que queramos. Da la posición (columna) de lo que se está escribiendo por impresora, y con ello se puede controlar (no del mejor modo) la impresión.

LPRINT Expresión

"Escribe en Impresora".

Sentencia compilable.

Es la encargada de que se escriba todo lo que sea menester en la impresora, produciendo copias documentales en papel del tipo que sea. A esto, la jerga informática a ultranza le dice "sacar una hard-copy" (sacar un copia dura). No es que el papel sea duro; más bien es que un papel dura más que una información en pantalla. Y como dura, es duro. Qué espanto.

Las reglas de uso de esta sentencia son análogas a las que se describen para su hermana-blanda PRINT, a excepción de todo lo referente a coordenadas de pantalla (capítulo 8 de la primera parte) que, naturalmente, en la impresora no han lugar.

```
20 ' ****** LOG ******
40 INPUT "NUMERO (0=FIN): ",A: IF A=0 THEN 70 ELSE X=A
60 PRINT "LOGARITMO NEPERIANO DE";A;" = ";LOG(A):GOTO 40
70 PRINT EXP(LOG(X));X

RUN <CR>
NUMERO (0=FIN): 521
LOGARITMO NEPERIANO DE 521 ES IGUAL A 6.25575
NUMERO (0=FIN): 22.6
LOGARITMO NEPERIANO DE 22.6 ES IGUAL A 3.11795
NUMERO (0=FIN): 0
 22.6  22.6
Ok

2 '****** LPOS ******
5 A$=STRING$(72,"A")
10 FOR I= 1 TO 100: IF LPOS(X)>30 THEN LPRINT CHR$(13)
30 LPRINT MID$(A$,I,1);: NEXT

RUN <CR>
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA      <---- Todo esto en impresora.
AAAAAAAAAAA
Ok

20 '****** LPRINT ******
30 '
40 'Ante todo, cuando en un programa usemos LPRINT
80 'hay que asegurarse que la impresora esta lista.
180 LINE INPUT "COSAS A IMPRIMIR (0 FIN): ",U$
190 IF U$="FIN" THEN END ELSE  LPRINT TAB(60-LEN(U$));U$
220 GOTO 180
230 'Las salidas producidas por la linea 190
240 'saldran por la impresora de la siguiente guisa:
250 '(supuesto que eso es lo tecleado en el LINE INPUT)
```

Direccion: JOSE LUIS GARCI
Produccion: Pulpitos Films S.J.
Guion: Javier Ignacio Albors
Realizacion: Ildefonso Elguerra & Emmanuel Praga

LPRINT USING Exp. Alf.; Expresión General

"Escribir en impresora utilizando Exp. Alf. en Expresión General"

Sentencia compilable.

(Rogamos al lector nos excuse de repetir las largas explicaciones dadas en el PRINT USING, ya que absolutamente todo lo que allí decimos es válido para el LPRINT USING, sin más diferencia que la salida de lo "printusado" tiene lugar en la impresora en vez de en la pantalla. Gracias).

LSET Var. Alf. = Exp. Alf.

Contracción del LEFT-SET ("Ponga Vd. a la Izquierda")

Sentencia compilable especialmente utilizada en tratamiento de ficheros.

Cuando vamos a grabar algo en un fichero de acceso directo (véase sentencia PUT) siempre hemos de hacerlo *preparando* a las variables que están definidas en la sentencia FIELD, que son las que en definitiva van a quedar grabadas. Esa preparación la realizan las sentencias LSET y RSET.

La primera (LSET) lo hace justificando por la izquierda el dato que figura en Exp. Alf. mientras que el RSET lo hace por la derecha.

En ambos casos, y con tales justificaciones, es la variable alfanumérica que acompaña al LSET o al RSET la que, mediante la sentencia que nos ocupa, queda cargada y debidamente acicalada para la grabación (para el PUT).

Existe una atípica utilización de estas dos sentencias fuera del manejo de ficheros. Si una variable alfanumérica la llenamos de espacios (en realidad la podemos llenar con lo que se nos antoje) una nueva asignación de esa variable a través de nuestras sentencias, produce el mismo efecto que en su uso en ficheros.

(Véanse ejemplos adjuntos y nota general a este respecto en el comienzo de esta segunda parte).

```
10 '******  LPRINT USING  ******
20 DEFDBL N
30 MA$="###,###,###.##"
40 INPUT N:IF N=0 THEN END
50 LPRINT USING MA$;N:GOTO 30

RUN <CR>
?12345
?123.2
?66666666
?0
Ok              (En la impresora aparecerà:)

                          12,345.00
                             123.20
                      66,666,666.00
```

```
10 '****** LSET ******
20 OPEN "R",1,"B:COSAS",14:FIELD #1,14 AS X$
40 FOR I=1 TO 5: READ A$
50 LSET X$=A$: PUT 1,I: NEXT I
57 '
60 FOR I=1 TO 5
70 GET 1,I: GOSUB 130
90 NEXT I: CLOSE
100 PRINT
110 LSET X$="POF":GOSUB 130: END
130 PRINT X$;LEN(X$):RETURN
140 DATA ESPIRITU,AMOR,RASPALINDERERAS
150 DATA LIS,IMPRESIONANTISIMO

RUN <CR>
ESPIRITU        14
AMOR            14
RASPALINDERERA  14
LIS             14
IMPRESIONANTIS  14

POF             14
Ok
```

MERGE Nombre de Programa

Merge = fundir, mezclar.

Comando puro.

Con el MERGE, llevamos a cabo la unión (más propiamente, el *mezclado)* de dos programas:

— El programa que en el momento del MERGE existe en memoria interna y

— el programa que se especifica en el MERGE.

El programa resultante, permanece vivo en memoria interna esperando a que Vd. haga uso de él.

Veamos cómo se lleva a cabo esa unión o mezclado.

Si en los programas no hay conflicto de líneas, es decir, si los números de línea de uno y otro programa están separados (no hay zonas comunes), se produce una simple y limpia yuxtaposición de uno y otro. No importa qué programa (si el "llamante" o el "llamado") tiene sus números de línea inferiores o superiores: un programa se "pega" al otro sin más problemas.

Cuando existen zonas comunes en la numeración de uno y otro puede ocurrir:

— Que aún siendo comunes las "zonas numéricas", no haya coincidencia de números de línea en concreto. En ese caso, los números, con sus instrucciones a cuestas, se entrelazan produciéndose una pacífica *mezcla* de numeraciones e instrucciones. No hay tampoco en este caso ningún tipo de conflicto.

— que haya números de línea coincidentes en ambos programas. En este caso —recuérdese— el programa llamado, el invitado, el que acompaña al comando, es el que prevalece. La línea en conflicto queda adjudicada no al del programa que inicialmente residía en la memoria sino al que acude a la llamada del MERGE. En ese caso, el contenido original de esa línea queda fulminado, machacado por el recién llegado.

```
10 '****** MERGE ******
20 'LINEAS DEL PROGRAMA "CLICLI"
30 'PARA QUE SE VEA COMO FUNCIONA EL MERGE
180 DATA CLIENTES MOROSOS, MOROSILLOS, MOROSUELOS,***

20 'PROGRAMITA DE SALIDA DE MENUS POR PANTALLA
40 FOR I=1 TO 25:PRINT:NEXT:'BORRA PANTALLA (CHAPU.)
50 RESTORE 180
60 READ A$:H=40-LEN(A$)/2: PRINT TAB(H);A$
80 PRINT TAB(H);STRING$(LEN(A$),"-")
90 FOR I=1 TO 20: READ A$: IF A$="***" THEN 150
120 PRINT TAB(H-1);I;A$: NEXT I:PRINT
150 PRINT "PULSE UNA DE LAS OPCIONES ";
160 A$=INPUT$(1)
180 DATA OPCIONES,PRIMERA,SEGUNDA,TERCERA,***

MERGE "CLICLI" <CR>
Ok
LIST <CR>
10 '****** MERGE ******
20 'LINEAS DEL PROGRAMA "CLICLI"
30 'PARA QUE SE VEA COMO FUNCIONA EL MERGE
40 FOR I=1 TO 25:PRINT:NEXT:'BORRA PANTALLA (CHAPU.)
50 RESTORE 180
60 READ A$:H=40-LEN(A$)/2: PRINT TAB(H);A$
80 PRINT TAB(H);STRING$(LEN(A$),"-")
90 FOR I=1 TO 20: READ A$: IF A$="***" THEN 150
120 PRINT TAB(H-1);I;A$: NEXT I:PRINT
150 PRINT "PULSE UNA DE LAS OPCIONES ";
160 A$=INPUT$(1)
180 DATA CLIENTES MOROSOS, MOROSILLOS, MOROSUELOS,***

RUN <CR>
                    CLIENTES MOROSOS
                    ----------------
                    1 MOROSILLOS
                    2 MOROSETES

PULSE UNA DE LAS OPCIONES
```

MID$ (Exp. Alf., Exp. Num1.[,Exp. Num2.])

MID$ (Exp. Alf., Exp. Num1.[,Exp. Num2.] = Exp. Alf2.

(Hemos preferido estudiar juntas ambas directivas. Distintas pero con un indudable parentesco).

MID = "por en medio de"

La primera es función alfanumérica. La segunda, sentencia. Ambas compilables.

La función (primer formato de la cabecera de página):

Cuando carece de la parte opcional (Exp. Num2.) quiere decir: "de la cadena alfanumérica Exp. Alf., quedate con la subcadena que empieza en el carácter número Exp. Num1. y llega hasta el final de Exp. Alf.".

Cuando se especifica el parámetro opcional (Exp. Num2.), lo traduciremos por "de la cadena Exp. Alf., toma la subcadena que comienza en el carácter número Exp. Num1. y a partir de él en adelante quédate con tantos caracteres como indique Exp. Num2.".

(Ver en capítulo 5 más detalles sobre esta función).

La sentencia (segundo formato de la cabecera de página):

Con ella sustituimos (*incrustamos*) en la cadena Exp. Alf1. que figura en el primer miembro, y en las posiciones en ella indicadas, el contenido de la Exp. Alf.2 que figura en el segundo miembro.

Si la cadena que *entra* tiene una longitud mayor que la que establecen los límites de la sentencia, entra "lo que puede".

Si la longitud de la cadena del segundo miembro es menor que esos límites, sólo quedan sustituidos los que corresponden a aquella.

La norma general es que la longitud de la cadena original *no sufre variación*. (En los ejemplos de la página de la derecha, se manejan todos los casos que puedan darse).

```
10 '******  MID$  ******
20 A$="PARANGARICUTIRIMICUARO"
22 ' PRIMERO MID$ - FUNCION.
30 FOR I=1 TO 5
40 PRINT MID$(A$,I,7)
50 NEXT I
60 PRINT MID$(A$,10)
70 PRINT MID$(A$,1,200)
80 PRINT MID$(A$,200,20)
90 PRINT MID$(A$,8,0)
100 '
110 ' Y AHORA MID$ - SENTENCIA
120 ' (SIEMPRE CON LA POBLACION MEXICANA)
125 PRINT A$
130 MID$(A$,8,6)="******": PRINT A$
150 MID$(A$,8,6)="..."
160 PRINT A$: ' (Solo entra lo que puede.)
170 MID$(A$,1,20)=""
180 PRINT A$: ' (No ha pasado nada.)
190 MID$(A$,20)="."
200 PRINT A$: ' (Solo entra ese puntito.)
210 MID$(A$,1)=STRING$(200,"#"): PRINT A$

RUN <CR>
PARANGA
ARANGAR
RANGARI
ANGARIC
NGARICU
CUTIRIMICUARO
PARANGARICUTIRIMICUARO


PARANGARICUTIRIMICUARO
PARANGA******RIMICUARO
PARANGA...***RIMICUARO
PARANGA...***RIMICUARO
PARANGA...***RIMICU.RO
######################
Ok
```

MKI$ Exp. Num. Entera
MKS$ Exp. Num. Simpre Precisión
MKD$ Exp. Num. Doble Precisión

Respectivamente, apócope de:

"Make Integer" = "Haz entero"
"Make Single" = "Haz de simple precisión"
"Make Double" = "Haz de doble precisión"

Funciones alfanuméricas de argumento numérico. Compilables.

Estas funciones (diseñadas específicamente para menejo de ficheros de acceso directo), convierten las expresiones numéricas que les acompañan en cadenas alfanuméricas de 2, 4 y 8 bytes de longitud, respecticamente.

Con la transformación de números en cadenas (se dice que los números se han *empaquetado*), podemos jugar con cantidades numéricas con una gran soltura.

La primera función de transformación (MKI$) permite empaquetar números enteros de simple precisión, es decir, comprendidos entre -32768 y 32767.

La segunda función (MKS$) permite, prácticamente, el empaquetamiento de cualquier número proporcionando una precisión de 6 cifras significativas.

La tercera (MKD$), por último, también permite lo mismo que la segunda pero disponiendo hasta de 16 cifras significativas.

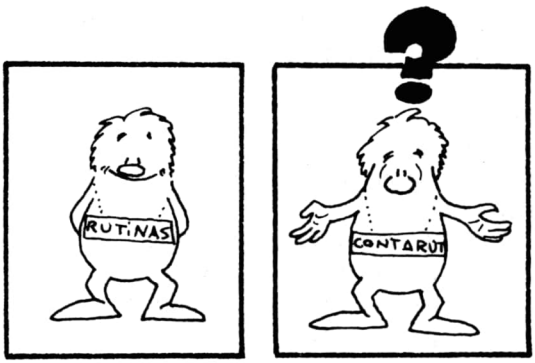
Estas funciones de empaquetamiento precisan de otras —inversas, de desempaquetamiento— que vuelven a traducir a números lo que por necesidades de programación habíamos tenido que convertir en alfanumérico. Estas funciones de conversión a lo numérico (que es tanto como decir a lo inteligible) son las CVI, CVS y CVD (véanse para completar este asunto).

En los ejemplos de la derecha, y de un modo sistemático se juega abundantemente con estos conceptos. Confiamos que con ellos se aclararán algunas dudas, mientras se desvanece del sufrido ánimo de nuestro lector (a estas alturas, seguramente ya muy templado) un comprensible brote de legítima impaciencia.

```
10 '****** MKI$  MKS$  MKD$ ******
20 DEFDBL V
30 OPEN "R",2,"B:MANGULOS",44
40 FIELD 2, 30 AS NOMB$, 2 AS MIN$,
           4 AS MED$,8 AS MAX$
50 '
60 ' GRABAMOS:
70 FOR I= 1 TO 2
80 LINE INPUT "NOMBRE: ",N$
90 LSET NOMB$=N$
100 INPUT "VALOR MINIMO: ",MIN
110 INPUT "VALOR MEDIO: ",MED
120 INPUT "VALOR MAXIMO: ",VMAX
130 LSET MIN$=MKI$(MIN)
140 LSET MED$=MKS$(MED)
150 LSET MAX$=MKD$(VMAX)
160 PUT 2,I
170 PRINT:NEXT I
180 '
190 ' LEEMOS
200 FOR I=1 TO 2
210 GET 2,I
215 PRINT CVI(MIN$);CVS(MED$);CVD(MAX$)
220 NEXT I
230 CLOSE:END

RUN <CR>
NOMBRE: MANGULORCIO
VALOR MINIMO: 1223
VALOR MEDIO: 34228
VALOR MAXIMO: 1654332

NOMBRE: MANGULIBRIDO
VALOR MINIMO: 7665
VALOR MEDIO: 85433
VALOR MAXIMO: 2564871

 1223   34228   1654332
 7665   85433   2564871
Ok
```

NAME Nombre de Fich.1 AS Nombre de Fich.2.

Traducimos: "Nombra al Fichero 1 Como Fichero 2"



Comando. Incluible en programa y compilable.

Sirve para cambiar el nombre de cualquier fichero o programa. El Fich.1 es el nombre antiguo (el que está todavía en vigor antes de aplicarle este comando) y Fich.2 el nuevo nombre que queremos que tenga.

Los posibles percances a encarar son:

— Que no exista Nombre de Fich.1. Como ya habrá adivinado el lector, se presentará ipso facto un elocuente "File not found" (Fichero no encontrado).

— Que ya exista Nombre de Fich.2. También es de presumir que la perspicacia del lector adelante el "File already exists" (Ya existe el fichero) que sin demora surgirá.

NEW

"Nuevo", "Renovado"

Comando incluible (asombroso) en programa. Cuando se llega a la línea en que figura este comando, el sistema pasa a modo control BASIC.

El efecto de este comando es el borrado de toda el área de usuario (menos las variables de error). Se trata, pues, de un comando terriblemente peligroso ya que con el tecleado de esas tres letras (seguidas de <CR>) puede producir un desastre verdaderamente dramático; puede echarnos a perder el trabajo de varias horas. (Ver comando SAVE).

```
NAME "B:PP.BAS" AS "B:PEPE.BAS"
Ok
NAME "A:MANOLO.BAS" AS "A:COSA.BAS"
File not found
Ok
NAME "MANUEL.BAS" AS "PEPE.BAS"
File already exists
Ok
LOAD "A:PEPE"
Ok
NAME "A:PEPE.BAS" AS "A:PEPE"
Ok
LOAD "A:PEPE"
File not found


10 '******  NEW  ******
20 Z$="PENELOPE": Z=999
25 PRINT Z$;Z
30 PRINT "VAMOS A PONER NEW"
40 NEW
50 PRINT Z$;Z

RUN <CR>
PENELOPE 999
VAMOS A PONER NEW
Ok

LIST <CR>
Ok

A=444:B>4                                        ***
Syntax error
Ok

NEW: PRINT A: PRINT ERR; ERL <CR>
 0
 2  65535
Ok
```

NOT Operando o Expresión lógico

Negación tajante, irrevocable.

Operador lógico compilable.

Es el único operador que no requiere más que un operando. Los demás operadores ya sean lógicos, relacionales o aritméticos, utilizan dos operandos entre los que se sitúa el operador. Por llevar la contraria, nuestro NOT la lleva hasta en eso.

En efecto, se trata de un operador que se limita a cambiar sin contemplaciones el sentido del operando o expresión lógica que va junto a él.

Si lo que acompaña al NOT es un número o una variable numérica (han de ser "enteros" uno u otra) el resultado de la operación lógica es el complemento "a doses" del número más uno. Dicho de otra forma, el NOT NUMERO es lo mismo que -(NUMERO + 1).

Y si lo que va junto al NOT es una expresión lógica, no hay más que dos posibilidades: O la expresión lógica es cierta (vale -1), con lo que su NOT es falso (vale 0), o la expresión lógica es falsa (vale 0) con lo que su NOT es cierto (vale -1). (Excúsenos el lector por semejantes galimatías).

OCT$ (Exp. Num.)

De "Octal".

Función alfanumérica de argumento numérico. Compilable.

Parecida a la función HEX$, la OCT$ convierte a base octal el valor de la Exp. Num. siempre que ésta sea un "entero" en base decimal, es decir, un número entero comprendido entre -32768 y +32767.

Debemos hacer observar que aquello en que queda transformado el número decimal, a pesar de la apariencia, no es propiamente un número sino una cadena alfanumérica.

(Como la HEX$, se trata de un función de uso sumamente restringido).

```
' ******  NOT  ******
PRINT NOT 0; NOT -1 <CR>
-1  0
Ok
PRINT NOT 1; NOT 30000; NOT -112 <CR>
-2 -30001 -111
Ok
PRINT NOT A; NOT B; NOT C=0 <CR>
-1 -1  0
Ok
X=1: Y=2: Z=3: PRINT NOT X>Y <CR>
-1
Ok
PRINT X<Y OR Z<Y <CR>
-1
Ok
M = NOT (X<Y OR Z<Y) <CR>
PRINT M <CR>
 0
Ok
PRINT NOT NOT X>Y; X>Y <CR>
 0  0
Ok
PRINT NOT NOT X<Y; X<Y <CR>
-1 -1
Ok

'******  OCT$  ******
PRINT OCT$(88) <CR>
 130
Ok
A=OCT$(8)*45 <CR>
Type mismatch
Ok
PRINT OCT$(22);VAL(OCT$(22)) <CR>
26 26
Ok
FOR I=1 TO 10:PRINT OCT$(I);" ";:NEXT <CR>
1 2 3 4 5 6 7 10 11 12 13
Ok
```

ON Exp. Num. GOTO Num. Lin.1, Num. Lin.2,...

ON Exp. Num. GOSUB Num. Lin.1, Num. Lin.2,...

"Sobre" ("Según el valor de")

Sentencia compilable.

Se puede utilizar en las dos formas que se dan en cabecera de página. Funciona como un IF múltiple, es decir, la primera expresión es equivalente a:

IF Exp. Num.=1 GOTO Num.Lin.1 ELSE IF Exp. Num.=2 GOTO Num. Lin.2...

Y la segunda a:

IF Exp. Num.=1 THEN GOSUB Num. Lin.1 ELSE IF Exp. Num.=2 THEN GOSUB Num. Lin.2...

Es decir, en ambos casos, con esta sentencia se produce una bifurcación del programa a las líneas que figuran a la derecha de la instrucción con arreglo al valor de la Exp. Num. Si ésta vale 1, el programa sigue en la *primera* línea que se especifica, si vale 2, el programa bifurca a la *segunda* línea, y así sucesivamente. Si el valor que toma la Exp. Num. es 0 o *es mayor* que el número de bifurcaciones, el programa sigue en la línea siguiente a aquella en que se encuentra el ON. Si ese valor es negativo o mayor de 255, lograremos un irremisible "Illegal function call" con abortado del programa.

Si el ON va acompañado de GOSUB y el valor de Exp. Num. está dentro del número de líneas de bifurcación, el programa ejecuta la rutina que corresponde al número del ON, retornando siempre a la línea siguiente a la del ON. Obsérvese que en este caso (ON con GOSUB) el programa *siempre* sigue en esa línea siguiente: bien por retorno de la rutina que corresponda, bien por entrada directa en esa línea cuando la Exp. Num. está fuera del número de bifurcaciones.

Señalemos por último que cuando el valor de la Exp. Num. no es entero, no se produce error: el sistema se encarga de redondear el valor que tiene la expresión (si la parte decimal no llega a 5 décimas, como se esa parte no existiera; y si es igual o mayor de esas 5 décimas, toma el entero inmediatamente superior).

```
10 ' *****  ON  *****
20 PRINT "1.- CONSULTAS"
30 PRINT "2.- MODIFICACIONES"
40 PRINT "3.- ALTAS"
50 PRINT "4.- BAJAS"
60 PRINT "5.- FIN"
70 INPUT N:IF N>5 OR N<1 THEN 20
80 ON N GOTO 100,1000,2000,3000
90 CLOSE:END
100 REM CONSULTAS
- - - - - - - - - - - - -
1000 REM MODIFICACIONES
- - - - - - - - - - - - -
- - - - - - - - - - - - -
```

```
10 ' ***** ON (sigue) *****
20 DATA PRIMAVERA,VERANO,OTONO,INVIERNO
30 DIM EST$(4)
32 'Cargamos el vector EST$
40 FOR I=1 TO 4:READ EST$(I):NEXT I
70 '
80 INPUT "ESTACION (0 A 4): ",E
90 ON E GOTO 200,300,400,500
95 IF E=0 THEN END ELSE GOTO 80
200 A$="DELICIOSO, AH!": GOTO 1000
300 A$="CALUROSO, UF!": GOTO 1000
400 A$="BASTANTE ACEPTABLE, EA!"
410 GOTO 1000
500 A$="HELADOR, AY!"
1000 PRINT "EN ";EST$(E);" EL TIEMPO ES ";A$
1010 GOTO 80
```

```
RUN <CR>
ESTACION (0 A 4): 2
EN VERANO EL TIEMPO ES CALUROSO, UF!
ESTACION (0 A 4): 1
EN PRIMAVERA EL TIEMPO ES DELICIOSO, AH!
ESTACION (0 A 4): 0
Ok
```

ON ERROR GOTO Num. Lin.

"En el caso de que aparezca un error vete a la línea Num. Lin.".

Sentencia compilable con serias reservas.

La misión de esta sentencia es la de que el programa pueda autocontrolar sus propios errores.

Si el error existe (naturalmente, un error detectable por el Basic), automáticamente el programa bifurca a la línea que aparece junto a la sentencia.

A partir de esa línea elegida por el programador, éste introducirá las instrucciones necesarias para que el error quede bajo su control y pueda obrar en consecuencia.

Aunque en apariencia se trata de una sentencia muy potente, la realidad es que adolece de bastantes inconvenientes a la hora de someter a compilación el programa (véase Anexo 2).

Si se desea que deje de funcionar la instrucción del ON ERROR, basta con poner en la propia rutina de control de errores la instrucción "ON ERROR GOTO 0" cosa que habría que hacer para todos aquellos errores que podriamos llamar normales (culpablemente normales, errores burdos de programación como "Sintax error", "Illegal function call", etc.).

```
10 '****** ON ERROR ******
20 ON ERROR GOTO 150
30 'Se supone que hasta la linea 4999 esta el programa
40 'sobre el que va a actuar la rutina de tratamiento
50 'de errores que comienza en la linea 5000.
60 '(Vease ejemplo en RESUME.)
70 '(Y vease tambien Tabla de Errores en Anexo.)
80 'Naturalmente, esto es un ejemplo de como
90 'puede organizarse una rutina de errores.
100 'No pretende agotar todas las posibilidades
110 'de un ON ERROR.
120 '(De cualquier modo insistimos en que
130 'a menos que no haya otro remedio, no se
140 'utilice esta sentencia.)
150 '
4999 '
5000 IF ERR=6 THEN 5080
5010 IF ERR=7 OR ERR=14 THEN 5100
5020 IF ERR=43 THEN 5140
5030 IF ERR=61 THEN 5160
5040 PRINT "Error de programacion, Nro.";ERR
5050 PRINT "en la linea";ERL
5060 PRINT "Revisa tu programa y corrigelo."
5070 CLOSE:END
5072 '
5080 PRINT "Numero demasiado grande."
5090 RESUME
5092 '
5100 PRINT "Nos hemos quedado sin memoria interna"
5110 PRINT "Tienes que revisar variables y vectores"
5120 PRINT "o ajustar el programa para que quepa."
5130 GOTO 5070
5132 '
5140 PRINT "Atencion: fichero saturado"
5150 PRINT "(Linea:";ERL;")":GOTO 5070
5152 '
5160 PRINT "Se ha llenado completamente el disco"
5170 GOTO 5150
```

OPEN Modo, [#] Nro. Fich., Nomb. Fich.[,Long. Reg.]

"Abrir un fichero"

Sentencia compilable.

Mediante el OPEN, se "abre" un fichero que es tanto como poder leer y/o grabar en él así como asignarle un número (Nro. Fich.) al que se referirá posteriormente cualquier operación que en él hagamos.

Hay tres "Modos" de abrir un fichero:

"O" (de "Out") para abrir un fichero secuencial en el que vamos a grabar. Atención: si existe ya otro fichero exactamente con el mismo nombre, quedará torpedeado e inutilizado por el nuevo.

"I" (de "In") id. id. en el vamos a leer. Si no existe ningún fichero con el nombre especificado (Nomb. Fich.), se producirá error («File not found»). Véase en Anexo 1 tabla con los errores más usuales.

"R" (de "Random") apertura de un fichero de acceso directo en el que vamos a leer o grabar.

El Num. Fich. (número de fichero) es un número entero sencillo por el que queda identificado el fichero. El Nomb. Fich. es el nombre del fichero en el que deseamos leer o grabar. La Long. Reg. (sólo en ficheros "R"), es la longitud de registro del fichero de acceso directo. Su valor por defecto es 128 bytes.

(Se sale del objetivo de este texto el desarrollo completo de esta secuencia. Nos remitimos a la nota general que a este respecto damos al comienzo de esta segunda parte).

```
'****** OPEN *****
OPEN "I", 1, "CUCUFATE" <CR>
File not found
Ok
OPEN "I", 1, "ADELFOS"
Ok
FOR I=1 TO 2: LINE INPUT #1,R$: PRINT R$: NEXT
Yo soy como las gentes que a mi tierra vinieron
soy de esa raza mora vieja amiga del sol
Ok
10 OPEN "O",4,"COSUCAS":
20 'Seguramente nos dara un Bad file number
70 'En interprete, el numero de ficheros que, por defecto
80 'se pueden abrir como maximo, son solo 3. En realidad
100 'podemos llegar hasta 15, pero solo si al al BASIC
110 'ponemos: MBASIC /F:15  o  MBASIC /F:7 por ejemplo.
140 CLOSE
145 '
146 'Otro asunto:
150 'Vamos a pasar el contenido de un fichero de acceso
160 'directo, llamado "AGENDA" (120 registros) de 64 bytes
180 'en el DRIVE A, a otro con el mismo nombre pero en el
190 'DRIVE B y reduciendo su longitud a 52 bytes.
220 OPEN "R", 1, "A:AGENDA",64
230 FIELD 1, 64 AS X$
240 OPEN "R", 2, "B:AGENDA",52
250 FIELD 2, 52 AS Y$
260 '
270 FOR R=1 TO 120
280 GET 1, R
290 LSET Y$=X$
300 PUT 2, R
310 NEXT R
320 CLOSE: END
360 '(Los 120 registros del A:, habran pasado
370 'al B:, reducidos de tamano, pero con
380 'todo lo demas, identico al fichero primitivo.)

RUN 220 <CR>
Ok
```

OPTION BASE N

"Opción de la base de vectores y matrices"

Sentencia compilable. Con ella decidimos con qué índice deseamos que comiencen a contar los elementos de *todos* los valores y matrices que vayan a aparecer en el programa. El valor del parámetro "N" que acompaña a la sentencia sólo puede valer 0 ó 1. Teniendo en cuenta que el valor por defecto (si no se da esta instrucción en el programa) es 0, resulta que la única forma práctica de que esa sentencia sirva para algo es poniendo:

OPTION BASE 1

Con ello, todo el programa considerará que no existen elementos "0" en las series uni o multidimensionales que el programa pueda incluir. Sirve, si se utiliza, para producir un ligerísimo ahorro de memoria interna.

OR

Conjunción adversativa "o" ("o esto o lo otro o lo de más allá").

Operador lógico compilable precedido casi siempre por un IF y que sirve para que se efectúe una acción siempre que se cumpla al menos *una* de las condiciones que aparecen separadas por los OR. (A lo largo de la primera parte de este texto, el lector podrá encontrar múltiples ejercicios de este importante operador).

OUT Exp. Num.1, Exp. Num.2.

De "Out" (fuera, hacia fuera).

Sentencia compilable. (Véase antes la función INP).

Con la función INP(P) conseguíamos que en ella entrase el valor de lo encontrado en la puerta lógica P. Mediante la sentencia OUT llevamos a la puerta cuyo número corresponde a la Exp. Num.1, el valor que tenga la Exp. Num.2. Ambos números deben ser del tipo "entero". El segundo, con procesadores de 8 bits, sólo podrá llegar a 255. Sentencia delicada y de cuyo uso recomendamos al lector primerizo se abstenga hasta que domine debidamente la materia.

```
OPTION BASE 1 <CR>
Ok
PRINT A(1): PRINT A(0) <CR>
 0
Illegal function call
Ok

5 '****** OR ******
10 A=5: B=14: C=10: PRINT 1 OR 1; 1 OR 2; 6 OR 7
30 PRINT A>B OR C>B; A=B OR B>C; A>(B+C) OR (A+C)>B
70 'Programa que coge un texto y cambia
80 'las mayusculas por minusculas.
90 LINE INPUT "ESCRIBE ALGO (O FIN): ",F$
100 IF F$="FIN" THEN END ELSE K$=""
110 FOR A=1 TO LEN(F$): ASCO=ASC(MID$(F$,A))
130 IF ASCO<65 OR ASCO>90 THEN K$=K$+CHR$(ASCO):GOTO 170
140 K$=K$+CHR$(ASCO OR 32)
142 'Tengase en cuenta que el byte '32' es '00100000'
143 'El 'OR' de ese byte con otro cualquiera
144 '"Levanta" el tercer bit y lo pone a '1'
145 'que es justo lo que caracteriza a las minusculas.
170 NEXT A
180 PRINT K$:GOTO 90
RUN <CR>
 1  3  7
 0 -1 -1
ESCRIBE ALGO (O FIN): CARLOS, MON AMOUR
carlos, mon amour
ESCRIBE ALGO (O FIN): AGENTE 007: A TI TE TUTEA TU TIA
agente 007: a ti te tutea tu tia
ESCRIBE ALGO (O FIN): FIN
Ok


10 ' ****** OUT ******
20 'Rutina de salida de un byte por la puerta 64
30 '(corresponde a una salida serie conectada a un plotter)
50 STATUS=INP(65) AND 4
60 IF STATUS=0 THEN 50 ELSE OUT 64,116
80 'El caracter cuyo ASC vale 116 sale al plotter.
```

PEEK (Exp. Num.)

De picotear, coger, pizcar.

Función numérica de argumento numérico compilable.

Esta potente función es capaz de introducir en ella nada menos que el contenido de lo que —atención— *en ese momento* haya en el byte número Exp. Num. de la memoria interna. No hay límite a la capacidad escudriñadora de esta función. Puede darnos lo que haya en *cualquier* octeto de la memoria interna. Incluso de aquella parte sagrada, intocable donde reside el sistema operativo. Dependiendo de la capacidad de la memoria interna del ordenador con el que estemos trabajando, así serán los límites permisibles para la Exp. Num., argumento de la función (que al menos, siempre será positivo).

POKE Exp. Num.1, Exp. Num.2.

Situar, colocar, poner.

Sentencia compilable.

Peligrosa y terrible sentencia por la que entramos directamente en la memoria interna para colocar en ella y en *cualquier* dirección (byte) de la misma (dado por Exp. Num.1.) *cualquier* valor comprendido entre 0 y 255 (Exp. Num.2). Este segundo parámetro tiene esa limitación (un octeto sólo puede tener valores entre 0 y 255). Como sucedía con el PEEK, el primer parámetro debe ser positivo y con un límite superior dependiente del espacio disponible en memoria interna concreta de la máquina con la que estemos.

POS (Arg. ficticio)

Posición.

Función numérica cuyo argumento puede ser lo que Vd. quiera pues es ficticio como él solo. Da la posición horizontal del cursor en pantalla en el mismísimo momento en que se está. La posición más a la izquierda es 1 y va creciendo hacia la derecha hasta el máximo que de la pantalla (normalmente 80). (Véase su hermana LPOS para aplicación a impresora).

```
X= PEEK(12323): PRINT X; CHR$(X)
 68 D
Ok
20 PRINT "METAMONOS EN LA MEMORIA INTERNA"
30 INPUT "DESDE: ",INIC:IF INIC=0 THEN END
50 INPUT "HASTA: ",FINAL
60 FOR I=INIC TO FINAL: A=PEEK(I)
80 IF A<32 OR A>122 THEN 140
90 PRINT CHR$(A);
100 Y$=INKEY$:IF Y$="" THEN 140
110 IF Y$=" " THEN PRINT:PRINT I
130 IF Y$=CHR$(27) THEN PRINT:GOTO 30
140 NEXT I: PRINT:GOTO 30

10 '****** POKE ******
20 'Mensajes de error BASIC en castellano.
30 INPUT "CAMBIAR: ",A$:IF A$="" THEN END
50 INPUT "PARA PONER A CAMBIO: ",B$
60 FOR I=1200 TO 20000:J$=""
70      FOR J=I TO I+LEN(A$)-1
80              J$=J$+CHR$(PEEK(J))
90      NEXT J:IF J$=A$ THEN 110
100 NEXT I
110 B$=LEFT$(B$+SPACE$(50),LEN(A$))
120 FOR K=I TO I+LEN(A$)-1
130 POKE K, ASC(MID$(B$,K-I+1))
140 NEXT K:GOTO 30

RUN <CR>                                        ***
CAMBIAR: Syntax error                           *
PARA PONER A CAMBIO: No comprendo               *
CAMBIAR:
Ok
PRONT J$ <CR>                                   ***
No comprendo
Ok

PRINT STRING$(40);POS(1):'****** POS
                                                 41
Ok
```

PRINT [LQUQ]

"Pinta, visualiza, escribe"

Sentencia compilable de todo uso.

Con PRINT hacemos aparecer en pantalla lo que Vd. quiera (iniciales de lo que en la expresión general se da entre corchetes).

Si no se añade nada al PRINT y éste aparece solitario en una instrucción, se produce un simple salto de línea. LQUQ puede consistir en variables y/o constantes numérica y/o alfanuméricas, funciones, etc. Las llamaremos "expresiones" en lo que resta de página.

Si LQUQ consta de varias expresiones separadas por comas, la salida por pantalla se produce en zonas separadas horizontalmente por una tabulación que suele ser de 14 posiciones. En este caso, si una de las expresiones a visualizar se pasa de la zona que le corresponde, automáticamente pasa a la siguiente (14 posiciones más a la derecha).

Si el separador que hay entre las expresiones es el punto y coma, éstas aparecerán en pantalla la una junto a la otra. Y más o menos juntas dependiendo del carácter alfanumérico o numérico de las expresiones. Si no se pone ningún separador entre ellas, las expresiones se visualizan como si hubiese un punto y coma entre ellas, cosa que tiene sus limitaciones.

El uso de esta sentencia es tan frecuente que, cuando se programa en intérprete, éste permite escribir el simple signo de interrogación como sustituto de la palabra "PRINT" (esto no es válido cuando el programa se desarrolla con un procesador o editor de textos, sistema legítimo de programar cuando vamos a trabajar sólo en compilado).

Sería imposible resumir aquí lo que se contiene en casi dos capitulos (3 y 8, éste completo) de la primera parte de este texto. A esos capítulos remitimos al lector si es que quiere completar su comprensión de esta no tan importante como frecuentísima sentencia.

```
10 '******  PRINT  ******
20 PRINT 1: PRINT -8: PRINT 4444
30 PRINT 1; -8; 4444: PRINT 1, -8, 4444
50 A=1: B=10: C=1000: PRINT B: PRINT C
70 PRINT MID$(STR$(B),2)
75 PRINT MID$(STR$(C),2)
80 PRINT C/B: PRINT A+B; C-A
90 A$="ACACIA": B$="PERITO": C$="MUCHO"
100 PRINT ACACIA; PERITO; MUCHO
110 PRINT A$; B$; C$: PRINT A$ + B$ + C$
130 PRINT A$;" ";B$;" ";C$
140 PRINT "SIN SALTO DE LINEA:";
150 FOR I=1 TO 10: PRINT;: NEXT
180 PRINT:PRINT "CON SALTO DE LINEA:"
190 FOR I=1 TO 3: PRINT: NEXT
220 PRINT "HASTA AQUI HA SALTADO"

RUN <CR>
 1
-8
 4444
 1 -8  4444
 1             -8              4444
 10
 1000
10
1000
 100
 11  999
 0  0  0
ACACIAPERITOMUCHO
ACACIAPERITOMUCHO
ACACIA PERITO MUCHO
SIN SALTO DE LINEA:
CON SALTO DE LINEA:



HASTA AQUI HA SALTADO
Ok
```

PRINT USING Exp. Alf.; Lista de Expresiones

"Escribe en pantalla utilizando Exp. Alf."

Esta sentencia se encarga de "pintar" (hacer aparecer en pantalla) la "lista de expresiones" utilizando el formato dado por Exp. Alf. A esta cadena-molde se le suele denominar «máscara».

Las expresiones a «formatear» pueden ser numéricas o alfanuméricas y si es más de una, deben ir separadas por punto y coma.

La Exp. Alf. puede ser de los siguientes tipos:

1.— Si la "Lista de expresiones" está formada por cadenas alfanuméricas.

1.1.— !: Sólo se toma(n) la(s) inicial(es) de la(s) cadena(s) que contiene(n) la Lista de Expresiones.

1.2.— *Espacios entre* \\ *:* Las cadenas se adaptan al número de espacios de la máscara más dos. Si una cadena es más larga que esa cantidad, los espacios que sobran son ignorados, como si no existieran. Si, al revés, la longitud de la máscara más dos es superior a la de la cadena, se visualizará la cadena seguida por el número de espacios en blanco que indique esa diferencia.

2.— Si la "Lista de expresiones" la componen expresiones numéricas:

2.1.— # : Una máscara formada por una cadena de "#", o de "#", comas (de millar) y/o punto (de decimal) hace que el valor numérico de la expresión se adapte a la máscara como a un molde. Si el número es inferior a ese molde se justifica el número por la derecha, o lo que es lo mismo, se rellena de espacios a la izquierda. Y si el número es superior a la capacidad de la máscara, el número aparece "enmascarado" (en la medida de lo posible), pero con un inesperado signo de % precediendo al número que, más o menos significa "te has pasado, es pequeña la máscara; o la hacer más grande o te seguiré sacando este signo que, dicho sea de paso, nada tiene que ver con un porcentaje".

```
PRINT USING "!";"ANTONIO";"BARTOLO";"CALIXTO"
ABC
Ok
PRINT USING "\     \";"Pepe";:PRINT "Manolo"
Pepe  Manolo
Ok
PRINT USING "\\";"Pepe";"Manolo";"Juanete"
PeMaJu
Ok

100 DEFDBL A
110 LINE INPUT "Mascara (CR=Fin): ",A$
120 IF A$="" THEN END
130 INPUT"Numeros a formatear (0=Fin): ",A,B
140 IF A=0 THEN 110
150 PRINT USING A$;A;B
160 GOTO 130
RUN <CR>
Mascara (CR=Fin): ###,###.##
Numeros a formatear (0=Fin): 85475.2,1235
 85,475.20  1,235.00
Numeros a formatear (0=Fin): 1234567,555.55
%1,234,567.00    555.55
Numeros a formatear (0=Fin): -12345,-1256.05
-12,345.00 -1,256.05
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): ######
Numeros a formatear (0=Fin): 123,1234
   123  1234
Numeros a formatear (0=Fin): 54321,52
 54321    52
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): #####.###
Numeros a formatear (0=Fin): -2,333.4
   -2.000  333.400
Numeros a formatear (0=Fin): -333333,4
%-333333.000    4.000
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): <CR>
Ok
```

Si la máscara formada por una cadena de « # » contiene una sola coma en alguna posición de la máscara, se entiende que se quiere producir una salida en forma de «millares» (con coma anglosajona en lugar del punto latino); no es preciso, pues, que la máscara presente la forma estricta que se desea ya que esa coma lleva al mismo fin. El punto decimal, en cambio, sigue la marcha por su cuenta, entendiendo que si el número no tiene los decimales que figuran en la máscra, aparecerán tantos ceros a la derecha como sea necesarios para completar dicha máscara.

Cuando el número a formatear es negativo, el signo aparecerá al comienzo, en la posición *anterior* a la definida por la máscara.

2.2.— +: Este signo, colocado al comienzo de una máscara numérica cualquiera (de acuerdo con 2.1.) hace aparecer el número con el formato de la máscara, *precedido* del signo + si el número es positivo o del signo - si el número es negativo. Si el signo + se sitúa al final de la máscara, éste aparecerá *al final* del número visualizado si es que éste es positivo. Si es negativo, será el signo - el que aparezca también al final.

2.3.— -: Si en la máscara se sitúa un signo - (menos) *al final* de la misma, aparecerá, al ejecutarse el PRINT USING un signo - *al final* del número visualizado. Si el número es positivo, no aparece nada.

2.4.— ** : Cuando se colocan dos asteriscos al comienzo de una máscara numérica, ésta aumenta en las dos posiciones dadas por los asteriscos, pero además, el número queda visualizado no con espacios en blanco sino con asteriscos. (Se trata de la clásica forma de proteger cantidades, de impedir que un desaprensivo añada dígitos inconfesables al comienzo de una cantidad, sean dólares, manguletes o pesetas).

```
10 '****** PRINT USING (Cont.) ******
20 '
30 DEFDBL A-B
40 LINE INPUT "Mascara (CR=Fin): ",A$
50 IF A$="" THEN END
60 INPUT"Numeros a formatear (0=Fin): ",A,B
70 IF A=0 THEN 40
80 PRINT USING A$;A;B
90 GOTO 60

RUN <CR>
Mascara (CR=Fin): ####,####
Numeros a formatear (0=Fin): 111111,22222
  111,111   22.222
Numeros a formatear (0=Fin): -200,19852.32
     -200   19,852
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): +###,###,###.##
Numeros a formatear (0=Fin): 88776655,5212
+88,776,655.00       +5,212.00
Numeros a formatear (0=Fin): -123.23,123
        -123.23         +123.00
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): #,##########.##+
Numeros a formatear (0=Fin): 3448577,-554433
  3,448,577.00+    554,433.00-
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): ####-
Numeros a formatear (0=Fin): 356,-11
 356   11-
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): ***,###,###.##
Numeros a formatear (0=Fin): 12345678,55.5
*12,345,678.00*********55.50
Numeros a formatear (0=Fin): -2,0
*********-2.00**********0.00
Numeros a formatear (0=Fin): 0,0
Mascara (CR=Fin): <CR>
Ok
```

PRINT USING Exp. Alf.; Lista de Expresiones (Cont.)

2.5.— ^^^^ : Rebuscada máscara la que utiliza, al final de ella, estos cuatro signos exponenciales. se utiliza en expresiones más o menos científicas (coma flotante). Los dígitos significativos se justifican por la izquierda, y el exponente se ajusta de acuerdo con la máscara. (Recomendamos al lector una de estas salidas a la aporía en la que seguramente se halla inmerso: o bien salta al párrafo siguiente si piensa que este tema no va con él —lo cual es bastante probable— o se enfrasca en los ejemplos de la derecha donde verá con claridad lo que para bien explicar exigiría una más larga y aburrida parrafada).

2.6.— _ (signo de subrayado): Permite agregar al formato algún carácter o cadena adicional, tanto al comienzo como al final de la máscara. Ver los ejemplos de la derecha.

2.7.— $$: Tampoco este caso está muy apropiado para quienes no trabajamos con dólares. Lo incluimos para completar estos ya largos comentarios relacionados con el PRINT USING. Funciona de modo análogo al "+", sólo que esta vez, es el signo más amado y odiado del mundo ($) el que sale precediendo al número formateado. Si en lugar de ese par de dólares ponemos al comienzo de la máscara ***, conseguiremos el doble efecto de este caso en que estamos junto con el de 2.4. O sea, que la parte no utilizada de la máscara se rellena por la izquierda con tantos asteriscos como espacios sobren, menos 1 reservado para el dólar, que no falte.

No nos resistimos a hacer un comentario final. El lector perspicaz habrá advertido que, a pesar del espacio que dedicamos al PRINT USING, esta sentencia no aparece subrayada en la cabecera de las páginas, lo cual es una forma de decir que, a nuestro juicio, no merece especial dignidad. Una larga experiencia con el BASIC nos dice que *siempre* hay una forma mejor de resolver la "salida bajo molde" de un dato, sea por pantalla o por impresora, por medio de rutinas de trabajo o de funciones definidas por el programador. Como ejemplo de lo que queremos decir, reflexiónese por ejemplo en el resultado práctico que proporciona el uso de la función, bastante universal ella, y definida así:

```
DEF FNJ$(N,Q)=RIGHT$(SPACE$(132)+STR$(N),Q)
```

```
'****** PRINT USING (Cont.) ******


PRINT USING "###.#^^^^";1234
 12.3E+02
Ok
PRINT USING "+###.##^^^^";444444!
+444.44E+03
Ok
PRINT USING ".###^^^^";123456789#
Ok


PRINT USING "_=######,#.#";6666.18
=  6,666.2
Ok
PRINT USING "***,###,###.##_ Pts.";4312
******4,312.00 Pts.
Ok
PRINT USING "_####,###.##_# Pts.";314259!
#314,259.00# Pts.
Ok


PRINT USING "$$#,#######";52500!
    $52,500
Ok
PRINT USING "$$#,###+";-1580;555.8
 $1,580-   $556+
Ok
PRINT USING "**$,###.00";15612
$15,612.00
Ok
PRINT USING "**$##,#######";-156000!
****-$156,000
Ok
```

PRINT # Nro. Fich., Lista de Expresiones

"Graba en el fichero abierto por el canal Nro. Fich., la Lista de Expresiones de la instrucción".

Sentencia compilable.

Es la responsable de la *grabación* de registros en un fichero secuencial. La grabación se efectúa con arreglo a un formato exactamente igual a como haríamos aparecer en pantalla (simplemente quitando) el signo "#") una lista de expresiones tal como se explica en la sentencia PRINT. Esto quiere decir que si por ejemplo, ponemos la lista de expresiones separadas por comas, en el disco o diskette se grabarán, además de los datos, los espacios necesarios para que esos datos queden debidamente tabulados. Aprovechamos el comentario para invitar al lector a que las expresiones de la lista a grabar vengan separados, no por comas sino por punto y comas.

(Pedimos nuevamente al lector vea la nota que en el comienzo de esta segunda parte hemos insertado respecto a todas las sentencias de tratamiento de ficheros).

PRINT # Num. Fich. USING. Exp. Alf.; Lista de Expresiones

«Graba en el fichero secuencial abierto con el número Num. Fich. y utilizando como máscara la Exp. Alf., la Lista de Expresiones».

Sentencia compilable.

Con ella, como con la anterior, grabamos datos en un fichero secuencial. Y lo hacemos con los datos "moldeados" de acuerdo a la máscara definida por Exp. Alf. cuya operatoria hemos visto en la sentencia PRINT USING.

Véanse los comentarios que hacemos en las dos sentencias (PRINT USING y PRINT #) y que son totalmente aplicables a esta sentencia que participa de la operatoria de las dos.

```
10 '******  PRINT #  y  PRINT #X USING  ******
20 OPEN "O", 2, "FICHSEC"
30 LINE INPUT "A GRABAR (F=Fin): ",R$
40 IF R$="F" OR R$="f" THEN 80
50 PRINT #2,R$:GOTO 30
70 '
80 MASC$="###,###,###.##":PRINT
90 INPUT "A GRABAR CON USING (F=Fin): ",R$
100 IF R$="F" OR R$="f" THEN 140
110 PRINT #2, USING MASC$; VAL(R$):GOTO 90
130 '
140 'Vamos a leer lo grabado.
145 PRINT
150 CLOSE:OPEN "I", 2, "FICHSEC"
160 IF EOF(2) THEN CLOSE:END
170 LINE INPUT #2,R$
180 PRINT R$:GOTO 160

RUN <CR>
A GRABAR (F=Fin): Y los pobres del mundo,
A GRABAR (F=Fin): una vez arriba,
A GRABAR (F=Fin): traeran prendidas cinco rosas
A GRABAR (F=Fin): proletarias sobre los luceros.
A GRABAR (F=Fin): f

A GRABAR CON USING (F=Fin): 1
A GRABAR CON USING (F=Fin): 123
A GRABAR CON USING (F=Fin): 12345
A GRABAR CON USING (F=Fin): 1234567
A GRABAR CON USING (F=Fin): f

Y los pobres del mundo,
una vez arriba
traeran prendidas cinco rosas
proletarias sobre los luceros.
          1.00
        123.00
     12,345.00
  1,234,567.00
Ok
```

PUT [#]Nro. Fich. [,Nro. Reg.]

"Pon (graba) en el registro Nro. Reg. del fichero abierto por el canal Nro. Fich. los datos que el LSET o el RSET haya preparado en la variables de campo definidas en la sentencia FIELD".

(Larga traducción de una de las instrucciones más breves —pero con más contenido— del BASIC).

Sentencia compilable. La única mediante la cual podemos grabar *directamente* el registro número X de un fichero abierto en modo "R" (Random, de acceso directo).

Respecto al *dónde* se graba la cosa está clara; lo repetimos: en el registro Nro. Reg. especificado al final de la instrucción del PUT.

Respecto al *qué* se graba. Echese una ojeada a las sentencias FIELD y LSET. En la ejecución del FIELD, quedan definidas (reservadas) las variables concretas que aparecen con esa sentencia. Y mediante el LSET (o RSET) cargamos esas variables con los datos que deseamos entren en el registro Num. Reg. que va con el PUT. Finalmente, la ejecución del PUT, hace que quede realmente grabado lo que pretendíamos.

Cuando un registro (es lo ordinario) está compuesto de varios campos —receptores/huecos/espacios— donde han de entrar los datos, y deseamos mantener todos los datos de un registro como estaban excepto, por ejemplo, uno solo cuyo valor queremos alterar, procederemos de la siguiente manera:

Con la sentencia GET, leeremos el registro a manipular. A continuación, mediante el LSET (o RSET) cargaremos la variable de FIELD que ha de sufrir modificación con el valor deseado. Un PUT subsiguiente hará grabar el registro con la modificación a cuestas (sin alteración para las variables intocadas).

Se puede utilizar el PUT sin especificar Nro. Reg. En tal caso, el registro por defecto es el *siguiente* al último que se haya manejado. Si no se ha tocado —GET o PUT— ninguno, la grabación se realiza en el *primer* registro del fichero.

(Ver nota inicial referente a sentencias de manejo de ficheros al comienzo de esta segunda parte).

```
10 ' ******  PUT  ******
12 '
20 ' Aprovechamos este ejemplo para grabar
30 ' un fichero de acceso directo
40 ' (longitud constante de los registros)
50 ' a partir de un fichero secuencial
60 ' (longitud de registro variable).
62 '
70 OPEN "I", 1, "FICHSEC"
72 'Abrimos para leer fichero secuencial.
80 OPEN "R", 2, "FICHDIR",64: 'Abrimos para
81 'grabar este fichero de acceso directo.
82 FIELD 2, 64 AS X$
84 N=0
90 IF EOF(1) THEN 250
92 N=N+1
100 LINE INPUT #1,A$
110 LSET X$=A$
120 PUT 2, N
130 GOTO 90
250 '
260 'VEAMOS COMO SE HA PORTADO EL PROGRAMA
265 '(Lo vamos a leer al reves.)
267 '
270 FOR I=N TO 1 STEP -1
280 GET  2, I
290 PRINT X$
300 NEXT I
310 CLOSE: END

RUN <CR>
  1,234,567.00
     12,345.00
        123.00
          1.00
proletarias sobre los luceros.
traeran prendidas cinco rosas
una vez arriba
Y los pobres del mundo,
Ok
```

RANDOMIZE [Exp. Num.]

Traduciremos por "aleatoriear" (horror) o mejor "iniciar aleatoriamente (al azar) un proceso en el que vamos a utilizar a su vez números aleatorios".

Sentencia compilable.

Su objetivo estriba en hacer más aleatoria (o, con mayor precisión, para hacer aleatorio desde su mismo comienzo), la serie de números que (aparentemente) al azar, el BASIC es capaz de generar.

Si no se utiliza, cada vez que se inicia (RUN) la ejecución de un programa que trabaje con la función RND, la serie de números "aleatorios" que vayan saliendo será siempre la misma.

Así resulta que si nuestra codicia nos lleva a hacer una programa para hacernos millonarios con la ruleta y no utilizamos el poderoso RANDOMIZE, los resultados de nuestra inconfesable investigación nos llevarán a un injustificado optimismo (o a un torpe pesimismo) ya que la serie de números que salga será siempre la misma a partir de cada RUN.

El número que corresponda a Exp. Num. debe ser "entero" (comprendido entre -32768 y 32767).

Si no hacemos uso de la opción entrecorchetada veremos aparecer el mensaje-entrada a la manera de un INPUT:

Random Number Seed (-32768 to 32767)?

En ese caso teclearemos la cifra que primero pase por nuestra mente con lo que conseguiremos que las cosas funcionen correctamente a continuación.

Fácil resulta comprobar que el número de "series aleatorias" diferentes que vamos a ser capaces de producir son tantas como las que expresan los límites que establece el "Random Number Seed", es decir, nada menos que 65636 series distintas. Ello hace que la aleatoriedad del sistema sea de una amplitud realmente abundante; podemos llevar a cabo simulaciones verdaderamente fecundas.

```
10 '******  RANDOMIZE  ******
20 FOR I=1 TO 10
30 A=INT(RND*100):PRINT A;
40 NEXT:PRINT
RUN <CR>
 24  30  31  51  5  78  49  36  98  90
Ok
RUN <CR>
 24  30  31  51  5  78  49  36  98  90
Ok

100 RANDOMIZE
110 FOR I=1 TO 10
120 A=INT(RND*100):PRINT A;
130 NEXT:PRINT
RUN <CR>
Random number seed (-32768 to 32767)? 25
 85  72  67  53  63  3  17  88  90  48
Ok
RUN <CR>
Random number seed (-32768 to 32767)? 44
 73  12  70  93  38  50  77  61  52  90
Ok

200 INPUT "Escribe algo: ",C$
220 FOR I=1 TO LEN(C$)
230 RR=RR+ASC(MID$(C$,I))
240 NEXT I
250 RANDOMIZE RR
260 FOR I=1 TO 10
270 A=INT(RND*100):PRINT A;
280 NEXT:PRINT
RUN <CR>
Escribe algo: PP
 23  5  2  39  60  53  47  24  42  15
Ok
Escribe algo: JJ
 64  36  52  89  98  32  83  57  24  2
Ok
```

READ Lista de Variables

"Lee". "Lee los datos que encuentres en DATA y carga con ellos las variables de la lista que te adjunto".

Sentencia compilable. Nuestro READ no tiene razón de ser sin su compañera DATA que recomendamos al lector que repase.

Cuando el programa tropieza con un READ, éste se apresura a buscar el *siguiente* dato y cargar con él la variable que va con el READ. Si es la primera vez que se ejecuta el READ, cámbiese "siguiente" por *primero* respecto al párrafo anterior.

Si la "lista de variables" contiene más de una, se van cargando ordenadamente, una tras otra, las variables que le acompañan con los sucesivos datos que existan en los DATA del programa. Si se han terminado esos datos aparecerá un "Out of DATA error". (Ver también sentencia RESTORE).

REM LQUQ

De "Remark" (Nota).

Sentencia compilable. Sustituible por el acento simple ('). No ocupa memoria cuando se compila.

No afecta en nada a la ejecución del programa. Exactamente como si no existiera. Detrás del REM se puede escribir LQUQ (lo que Vd. quiera), los comentarios que mejor le sirvan para que el programa no se convierta en un trabalenguas indescifrable cuando Vd. vuelva a él en el futuro.

Si una línea de instrucciones tiene un REM en su comienzo, *toda* la línea completa es ignorada, aún cuando existan instrucciones "ejecutables". Si el REM está en medio de una línea, se ejecutarán las instrucciones que haya *antes* del REM, pero no las que pueda haber tras él.

(Debemos dejar constancia aquí de una experiencia notable. Si un mismo programa es confeccionado por un germano-sajón y por un latino, podemos apostar que el de éste último dispone como mucho de la mitad de "instrucciones REM" que el del primero. Resulta paradójica y sorprendente la pereza que solemos tener para algo que, en el fondo sólo sirve para ahorrarnos trabajo).

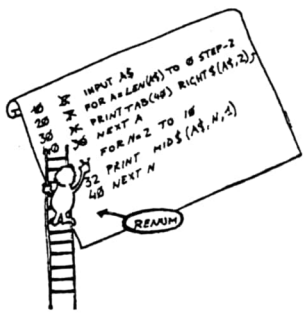
```
10 ' ******  READ  ******
15 '
20 DATA CERO,UNO,DOS,TRES,CUATRO
25 CINCO,SEIS,SIETE,OCHO,NUEVE
30 RESTORE 20
32 '
40 A$=INPUT$(1)
50 IF A$="F" THEN END
55 '
60 FOR A=0 TO VAL(A$)
70 READ NUM$
80 NEXT A
85 '
86 IF A$="0" OR VAL(A$)<>0 THEN 90
87 NUM$="CARACTER NO NUMERICO"
90 PRINT "HAS PULSADO UN ";NUM$
100 GOTO 30

RUN <CR>
HAS PULSADO UN SIETE
HAS PULSADO UN CINCO
HAS PULSADO UN OCHO
HAS PULSADO UN SEIS
HAS PULSADO UN CARACTER NO NUMERICO
HAS PULSADO UN CERO

Ok          <--------- Hemos pulsado una "F"


10 '******  REM  ******
20 REM PROGRAMA DE EMULACION BIDIRECCIONAL
30 REM REALIZADO BAJO CONTROL DE LA NASA
40 ' EN EL QUE SE PONE DE MANIFIESTO
50 ' LA IDENTIDAD OPERATIVA
60 REM ENTRE LA INOFENSIVA SENTENCIA REM
70 ' Y SU EQUIVALENTE NO MENOS INOFENSIVO '
80 REM FIN DE PROGRAMA (SIGUEN 7 COPYRIGHTS)

RUN <CR>
Ok
```

RENUM [[Nueva Lin][,[Antig. Lin.][Incremento]]]

De "Renumber" (Renumerar).

Comando puro. Con él se renumera un programa completo o una parte del mismo.

Partimos de la base de que en memoria interna hay un programa cuya primera línea es *1er Nro. Lin.* En la expresión general del RENUM, "Nueva Lin." significa el nuevo número de línea que queremos que tenga aquélla por la que deseamos comience la renumeración. "Antig. Lin.", es el número de línea antiguo (anterior al RENUM) en el que va a dar comienzo esa renumeración. Y por último, el "Incremento" es el valor del salto numérico entre líneas.



Los valores por defecto de estos parámetros son:

— Del "Incremento": 10
— De «Antig. Lin»: la primera que hubiere (1.er Nro. Lin.).
— De "Nueva Lin.": 10

La ausencia de un parámetro significa la adopción del valor por defecto del mismo y se expresa escribiendo comas sin más. La ausencia total de parámetros (RENUM a secas) produce la renumeración del programa a partir de la línea 10 y con saltos entre líneas de 10 en 10.

Si se intenta un absurdo (vease ejemplo a la derecha), se nos recompensa con un "Illegal function call". En cualquier RENUM (sin error), las referencias que haya dentro de las instrucciones del programa a otros números de línea (mediante GOTO, GOSUB, ON, etc.) quedan automáticamente adaptadas a la nueva numeración. Cuando los números de línea de esas referencias no existan, tales números permanecerán inalterados aunque, surgirá un mensaje (una vez afectuada la renumeración) de aviso de dicha inexistencia.

```
10 '******  RENUM  ******
12 N=0
15 N=N+1
17 IF N>=10 THEN 122
22 GOTO 15
122 PRINT "SANSEACABO"

RENUM <CR>
Ok
LIST <CR>
10 '******  RENUM  ******
20 N=0
30 N=N+1
40 IF N>=10 THEN 60
50 GOTO 30
60 PRINT "SANSEACABO"
Ok

RENUM 500,30,5 <CR>
Ok
LIST <CR>
10 '******  RENUM  ******
20 N=0
500 N=N+1
505 IF N>=10 THEN 515
510 GOTO 500
515 PRINT "SANSEACABO"
Ok

RENUM 21,20,13 <CR>
Ok
LIST <CR>
10 '******  RENUM  ******
21 N=0
34 N=N+1
47 IF N>=10 THEN 73
60 GOTO 34
73 PRINT "SANSEACABO"
Ok
```

RESTORE [Nro. Lin.]

"Restaura, vuelve a empezar".

Sentencia compilable.

Tiene su aplicación sólo en programas en que existan las sentencias READ y DATA.

Las sentencias DATA son inoperantes, bellas durmientes a la espera del beso del READ. Y cada READ, una vez ejecutado, hace avanzar al apuntador del DATA un lugar, señalando al siguiente dato.

La sentencia RESTORE tiene la virtud de llevar al apuntador a los datos que figuren tras la sentencia DATA que queramos; exactamente a los de la línea "Nro. Lin." que va junto al RESTORE.

Si no se especifica nada, el valor por defecto del RESTORE es el del DATA cuyo número de línea es el más bajo dentro del programa.

En cualquier caso, una vez ejecutado un RESTORE, el primer READ que entre un funcionamiento siguiendo la secuencia lógica del programa, leerá el primer dato que exita en la línea que va junto al RESTORE (o simplemente el "primero" del programa si el RESTORE se encuentra solo).

(Para más información, véanse las sentencias READ y DATA, compañeras de fatiga de nuestro RESTORE. Y también, ya puestos, proponemos al lector que eche una ojeada al capítulo 3 de la primeral parte de este texto).

```
RESET <CR>
Ok

10 '****** RESTORE ******
20 DATA 0,31,59,90,120,151,181,212,243,273,304,334,365
30 DATA ENERO,FEBRERO,MARZO,ABRIL,MAYO,JUNIO,JULIO
40 DATA AGOSTO,SEPTIEMBRE,OCTUBRE,NOVIEMBRE,DICIEMBRE
50 D=0:B=0:INPUT "Dias transcurridos del ano (CR=Fin): ",D$
60 V=0:D=VAL(D$):IF D=0 THEN END ELSE IF D>365 THEN 50
70 M=0:RESTORE 20
80 M=M+1:READ A:IF D<=A THEN 90 ELSE B=A:GOTO 80
90 RESTORE 30
100 FOR I=1 TO M-1:READ M$:NEXT
110 DIA=D-B:IF V THEN 150 ELSE PRINT "Hoy es";DIA;"de ";M$
120 INPUT "Dias Vencimiento (CR=FIN): ",V:IF V=0 THEN 50
130 D2=D:D=D+V:D1=D:IF D>365 THEN D=D-365
140 GOTO 70
150 PRINT "Vencimiento: ";DIA;"de ";M$;
160 D=D2:IF D1<366 THEN PRINT:GOTO 120
170 PRINT " del ano siguiente.":GOTO 120
RUN <CR>
Dias transcurridos del ano (CR=Fin): 188
Hoy es 7 de JULIO
Dias Vencimiento (CR=FIN): 90
Vencimiento:  5 de OCTUBRE
Dias Vencimiento (CR=FIN):  <CR>
Dias transcurridos del ano (CR=Fin): 300
Hoy es 27 de OCTUBRE
Dias Vencimiento (CR=FIN): 180
Vencimiento:  25 de ABRIL del ano siguiente.
Dias Vencimiento (CR=FIN): <CR>
Dias transcurridos del ano (CR=Fin): 32
Hoy es 1 de FEBRERO
Dias Vencimiento (CR=FIN): 30
Vencimiento:  3 de MARZO
Dias Vencimiento (CR=FIN): 400
Vencimiento: 8 de MARZO del ano siguiente.
Dias Vencimiento (CR=FIN): <CR>
Dias transcurridos del ano (CR=Fin): <CR>
Ok
```

RESUME [Algo]

(Atención el verbo "resume" en inglés, nada tiene que ver con "resumir", significa exactamente "REANUDAR, Recomenzar).

Sentencia compilable con reparos (misma situación que ON ERROR).

Esta sentencia, que debe ser colocada en la rutina de detección de errores, (debe, por tanto existir en el programa un ON ERROR y una rutina de tratamiento de errores) hace que se reanude la ejecución del programa (una vez que el error ha sido debidamente tratado-regañado-desemponzoñado).

- En la misma instrucción en la que se produjo el error si el RESUME está solo o con un pobre 0 (RESUME o RESUME 0).
- En la instrucción siguiente a aquélla en que se produjo el error, si el "Algo" que va junto al RESUME es la palabra NEXT (=siguiente); es decir, RESUME NEXT.
- En la línea especificada por el número que (si existe) acompaña al RESUME; es decir, RESUME Nro. Lin.

La sentencia RESUME sólo tiene sentido (insistimos) dentro de la porción de programa designado previamente por la sentencia ON ERROR GOTO Num. Lin. Si no existe esta sentencia, la RESUME queda inactiva, perdida, huérfana, estéril.

Queremos poner de manifiesto que cuando un programa está dotado de ON ERROR y de RESUME, hay que decírselo al compilador (no es el momento de explicarlo). Y un programa compilado en esas condiciones, respecto al mismo sin tratamiento de errores, aparte de requerir (gastar) bastante más memoria, trabaja peor (más lentamente), por lo que nos resistimos a animar al lector para que emplee esta sentencia, como no sea en la primera fase de su aprendizaje, mientras no pase de operar en intérprete.

```
5  '******  RESUME  ******
10 ON ERROR GOTO 1000
20 DEFINT N:I=2222:PINT I:'FLAGRANTE ERROR DE SINTAXIS
105 N=30000
110 N=N+1000:PRINT N;
117 IF N>35000! THEN PRINT:GOTO 130
120 GOTO 110
130 NEXT J
140 IF N>40 THEN 200
150 PRINT "SEGUIMOS:":DEFDBL N:GOTO 105
200 PRINT"PERO CONTINUAMOS, RESUELTAMENTE."
210 ERROR 129
220 END
1000 ON ERR GOTO 1100,1200,1300,1300,1300,1600
1008 IF ERR<>129 THEN 1300
1010 PRINT "HEMOS FORZADO AL PROGRAMA A QUE INCURRA"
1012 PRINT "EN UN EXTRANO ERROR,  A VER QUE PASA (nada)."
1016 RESUME 220
1100 PRINT "HAY UN NEXT SIN UN FOR PREVIO EN LINEA";ERL
1120 RESUME 200
1200 PRINT "VERGUENZA: ERROR DE SINTAXIS EN LINEA";ERL
1230 RESUME 105
1300 RESUME 0
1600 PRINT:PRINT "AHORA, POR SI FUERA POCO, LA VARIABLE N, "
1610 PRINT "QUE ESTA DEFINIDA COMO ENTERA,"
1620 PRINT "HA REBASADO SU CAPACIDAD, EN LA LINEA";ERL
1630 RESUME 150
RUN <CR>
VERGUENZA: ERROR DE SINTAXIS EN LINEA 20
 31000  32000
AHORA, POR SI FUERA POCO, LA VARIABLE N,
QUE ESTA DEFINIDA COMO ENTERA,
HA REBASADO SU CAPACIDAD, EN LA LINEA 110
SEGUIMOS:
 31000  32000  33000  34000  35000  36000
HAY UN NEXT SIN UN FOR PREVIO EN LINEA  130
PERO CONTINUAMOS, RESUELTAMENTE.
HEMOS FORZADO AL PROGRAMA A QUE INCURRA
EN UN EXTRANO ERROR, A VER QUE PASA (nada).
Ok
```

RIGHT$ (Exp. Alf., Exp. Num.)

De Right = Derecha, por la derecha.

Función alfanumérica de argumento mixto (alfanumérico y numérico), compilable.

Esta función coge, de la cadena Exp. Alf. y por la derecha un número de caracteres igual al valor numérico de la Exp. Num. Es una función muy parecida a LEFT$ (véase) y como ésta, puede considerarse un caso particular de la función MID$, mucho más general y de más frecuente uso que estas dos (RIGHT$ y LEFT$).

Como las otras dos funciones de tratamiento de cadenas sólo se produce error si la Exp. Num. tiene valor negativo. En el resto de los casos:

- Si la Exp. Alf. contiene la cadena nula, sea cual fuere el valor de Exp. Num., esta función producirá también la cadena nula.
- Si el valor de Exp. Num. es mayor que la longitud (LEN) de Exp. Alf., obtendremos, al aplicar esta función, exactamente lo mismo que Exp. Alf. contenía.

(Ver función LEFT$ y capítulo 5 de la primera parte).

RND [(Exp. Num.)]

Consonantes de "Random" ("Al azar, aleatorio").

Función numérica compilable.

Esta función (apoyada o no por la sentencia RANDOMIZE), genera un número aleatorio (pseudo-aleatorio, cuasi-aleatorio) comprendido entre 0 y 1. Cuando decimos "entre 0 y 1", atención, no incluimos ni el 0 ni el 1 exactos.

Si no hacemos uso de Exp. Num., es decir, si ponemos sólo RND el resultado es el mismo que si lo acompañamos por cualquier número positivo. Si la Exp. Num. tiene valor cero, el número creado es el *último* que la función RND generó. Y si es negativo, se produce una especie de RESTORE de la tabla de números aleatorios al primero de la serie. (Véase capítulo 6 de la primera parte para mayor información).

```
'******   RIGHT$   ******
A$="SERAN CENIZA MAS TENDRAN SENTIDO"
PRINT RIGHT$(A$,7)
SENTIDO
Ok
PRINT RIGHT$(A$,12)
DRAN SENTIDO
Ok
PRINT RIGHT$(A$,0)

Ok
PRINT RIGHT$(A$,250)
SERAN CENIZA MAS TENDRAN SENTIDO
Ok

V$="DEJANDO MI CUIDADO ENTRE LAS AZUCENAS OLVIDADO"
D=8: PRINT RIGHT$(V$,D): PRINT MID$(V$,LEN(V$)-D+1)
OLVIDADO
OLVIDADO
Ok
D=27: PRINT RIGHT$(V$,D): PRINT MID$(V$,LEN(V$)-D+1)
ENTRE LAS AZUCENAS OLVIDADO
ENTRE LAS AZUCENAS OLVIDADO
Ok


'******  RND  ******
PRINT RND;RND;RND
 .770562  .098734  .518358
Ok

20 FOR I=1 TO 8
30 PRINT INT(RND*1000)
40 NEXT I
50 '(Ver RANDOMIZE y Primera Parte.)

RUN <CR>
 770  98  518  884  317  235  921  167
Ok
```

RUN [Nombre de Programa][Nro. Lin.]

"Run" = Correr, rodar, ejecutar, poner en marcha.

Comando, incluible en programa, compilable.

Sin la parte opcional, con el RUN a secas seguido de CR, este comando pone en marcha el programa que haya en memoria en ese momento comenzando por la línea cuyo número sea más bajo.

Si se añade un Nombre de Programa, pueden suceder muchas cosas:

- Que no se exprese drive (A:, B:, etc.). El programa será buscado en el drive por defecto, es decir, en aquél que estuvieramos al cargar el intérprete Basic.
- Que se especifique un drive. El programa sera buscado en ese drive.
- En cualquiera de los casos anteriores, puede ser que exista o que no exista tal programa. Si existe, ese programa pasará de la memoria externa (disco o diskette) a la interna y empezará su ejecución como con el RUN a secas.
- Si no existe el programa en cuestión se presentará el conocido "File not found" explícito e inequívoco.

En todo caso lo primero que tiene lugar es un NEW (una total limpieza de la memoria interna), junto con un cierre de ficheros.

Si se utiliza la opción "Num. Lin.", la ejecución comienza en ese número de línea, ojo, previa la misma limpieza de memoria antes mencionada.

(Si queremos que se ejecute el programa manteniéndose los valores de las variables existentes en memoria interna, pondremos "GOTO Nro. Lin.", siendo este Nro. Lin. cualquiera de los que tenga el programa, incluso el primero).

Cuando el "RUN Nombre de Programa" está incluido dentro de otro programa, se produce la búsqueda y ejecución del programa especificado, desapareciendo el valor que tuviesen las variables en el primero. Este comportamiento es semejante al del "CHAIN" (véase), salvo en que con el RUN no existe ninguna posibilidad de paso de valores de variables entre los programas.

```
'******  RUN  ******

RUN "B:PRIMOS" <CR>
 1  2  3  5  7  11  13  17  19  23  29
Ok

RUN "A:BOBADA" <CR>
 1  4  9  16  25  36  49  64  81  100  121
Ok

LIST <CR>
10 DIM A(20)
20 FOR I=1 TO 11
30 A(I)=I*I
40 NEXT I
50 FOR I=1 TO 11: PRINT A(I);:NEXT I

GOTO 50 <CR>
 1  4  9  16  25  36  49  64  81  100  121
Ok

RUN 50 <CR>
 0  0  0  0  0  0  0  0  0  0
Subscript out of range in 50
Ok

NEW <CR>
Ok
10 A=10
20 RUN "A:BOBADA"

RUN <CR>
 1  4  9  16  25  36  49  64  81  100  121
Ok

PRINT A <CR>
 0
Ok
```

SAVE Nombre Programa [, Letra]

De "Save" = "Salvar", "salvaguardar".

Comando incluible (!!) en programa. Irrelevante en compilación.

Con este comando grabamos en disco-diskette el programa BASIC que pueda haber en memoria interna. La memoria interna, con todas sus excelencias, es volátil, se borra completamente cuando cesa la corriente. Esto significa que estamos en precario respecto a la disponibilidad futura del programa que estemos elaborando, a menos que los "guardemos" (que lo "grabemos", que lo "salvemos") en algún sitio de donde podamos "recuperarlo" incluso después de «apagado» el ordenador. Ese sitio es el disco o el diskette de nuestra máquina. Y la "palabra mágica" capaz de llevar a cabo tan importante misión es el comando que nos ocupa: SAVE.

Por " Nombre de Programa" entendemos cualquier combinación de letras y números entre 1 y 8. Este nombre puede venir precedido por la identificación del disco donde deseamos que se produzca su grabación (una letra seguida de "dos puntos").

Ojo: si *ya existe* en el disco un programa con el mismo nombre ése queda completamente machacado y sustituido por el recién llegado que el "SAVE" trae de la mano.

Podemos salvar un programa de tres diferentes modos:

- Sin apéndice ninguno (sin hacer uso de la opción [Letra] que figura en la expresión general de cabecera). En ese caso el programa queda grabado en forma binaria, comprimida, sólo comprensible por el intérprete BASIC. Utilizaremos este modo siempre que el programa vaya a ser ejecutado en intérprete.
- Si al final del SAVE ponemos, *P* (P de "proteger"), el programa queda "protegido", incapacitado para que sobre él pueda actuar el comando LIST y por tanto a la indiscreción de algún curioso. (Protección, digámoslo, no excesivamente rigurosa).
- Por último (la más importante), si el apéndice es, *A*, el programa queda salvado en ASCII, es decir, en forma de texto normal. Es el modo de salvar cuando el programa va a ser compilado. El compilador de Microsoft exige que el programa esté en esta forma, la cual es idéntica a la que adoptaría si el programa lo hubiésemos preparado mediante un editor o un procesador de textos.

```
' ******  SAVE  ******
LOAD "VEAMOSLO" <CR>                          ***
File not found
Ok

10 PRINT "Este Programa non fa niente"  ***
SAVE "VEAMOSLO" <CR>                          ***
Ok
LOAD "VEAMOSLO" <CR>                          ***
Ok
LIST <CR>                                     ***
10 PRINT "Este programa non fa niente"
Ok

LOAD "SECRETO" <CR>                           ***
Ok
LIST <CR>                                     ***
10 PRINT "IDENTIFICATE, AMIGO ";
20 ID$=INPUT$(5):IF ID$<>"XYOO7" THEN 10
30 '
40 'Has escrito la clave correcta por lo que
50 'el programa puede seguir tranquilo.
55 '
70 ' XIU, XIU, SNIF, SNIF, BLA, BLA
80 ' MUCHOS Etceteras en los que
90 ' hasta la linea 999 volcamos cantidad
95 ' de valiosisimos secretos.
100 '
1000 ' (Pues vaya un programa "secreto"
1010 ' en el que el LIST nos lo da todo.)
Ok

SAVE "SECRETO",P <CR>                         ***
Ok
LOAD "SECRETO" <CR>                           ***
Ok
LIST <CR>                                     ***
Illegal function call
Ok
```

SGN (Exp. Num.)

Consonantes de "Sign" (Signo aritmético).

Función numérica de argumento numérico, compilable.

Muy sencilla función. Según sea su valor, el resultado de esta función es:

1 si Exp. Num. es positivo
0 si Exp. Num. vale cero
-1 si Exp. Num. es negativo.

SIN (Exp. Num.)

De "Sinus", seno.

Función numérica de argumento numérico, compilable.

Dá el valor del seno trigonométrico del ángulo *en radianes* dado por Exp. Num.

El valor que saca está en "simple precisión", es decir, propocionando 6 cifras significativas.

SPACE$ (Exp. Num.)

"Espacio", "Espaciado", "Blancos"

Función compilable alfanumérica de argumento numérico.

Proporciona una cadena de tantos blancos (espacios) como indica el valor de Exp. Num.

Exp. Num. puede tomar cualquier valor comprendido entre 0 y 255. Si el argumento tiene valor vero, la función queda cargada con la cadena nula (""). Si el argumento es negativo, salta el típico Illegal function call. Son equivalentes las dos expresiones:

SPACE$(X) y STRING$(X,32)

```
' ****** SGN ******
A=4: PRINT SGN(A) <CR>
 1
Ok
B=-1234:PRINT SGN(B) <CR>
-1
Ok
C=0:PRINT SGN(C) <CR>
 0
Ok

10 '****** SIN ******
20 FOR I=1 TO 1.4 STEP .1
30 PRINT SIN (I)
40 NEXT I
RUN <CR>
 .841471
 .891207
 .932039
 .963558
Ok

10 ' ***** SPACE$ *****
20 'ESTE PROGRAMA RELLENA CON ESPACIOS EN BLANCO
30 'A LA DERECHA HASTA COMPLETAR UNA LONGITUD DESEADA
50 INPUT "Longitud (0=Fin): ",L:IF L=0 THEN END
70 INPUT "Dato a rellenar de blancos: ",D$
80 IF LEN(D$)>L THEN 50
90 DATO$=D$+SPACE$(L-LEN(D$)):PRINT DATO$;LEN(DATO$)
100 GOTO 50

RUN <CR>
Longitud (0=Fin): 22
Dato a rellenar con blancos: PACORRO
PACORRO                22
Longitud (0=Fin): 22
Dato a rellenar con blancos: PACO
PACO                   22
Longitud (0=Fin): 0
Ok
```

SPC (Exp. Num.)

Consonantes de SPACE.

Función compilable.

Semejante a SPACE$ pero más limitada que ésta.

Esta función, no cargable sobre variables de ningún tipo, tiene su aplicación exclusivamente en salidas por pantalla o impresora y no añade nada a su hermana mayor SPACE$ pues tiene las mismas limitaciones en cuanto a su longitud.

SQR (Exp. Num.)

Abreviatura de "Square root" (raiz cuadrada).

Función numérica de argumento numérico, compilable.

Facilita el valor de la raiz cuadrada de Exp. Num. La única limitación de esta función es que el argumento no puede ser negativo (el ordenador no sabe de números imaginarios).

STOP

Parar, interrumpir.

Sentencia compilable.

El encuentro con esta sentencia interrumpe sin contemplaciones el programa en ejecución situando al operador en "modo control".

Cuando el programa está en intérprete, al llegar a un STOP, aparece un mensaje en pantalla:

Break in line xxxx

en la que xxxx indicará el número de línea en que ha quedado abortado el programa (que será precisamente aquélla en la que se encuentra la sentencia STOP). Se diferencia el STOP del END —aparte de que el END no facilita ese número de línea en que el programa muere— en que el STOP deja *abiertos* los ficheros que pudieran encontrarse abiertos en el momento de la interrupción, cosa que no ocurre con el END, que lo cierra todo.

```
10 '****** SPC ******
20 FOR I= 0 TO 10 STEP 4: PRINT SPC(I);"Carlota":NEXT
60 A=SPC(10): 'Va a dar un error como la copa de un pino.
RUN <CR>
Carlota
    Carlota
        Carlota
Syntax error in 60

100 '****** SQR ******
110 INPUT A#: IF A#=0 THEN END
120 PRINT A#;SQR(A#);
130 B#=SQR(A#):PRINT A#;B#:GOTO 110
RUN <CR>
? 400
 400  20  400  20
? 625
 625  25  625  24.99999809265137
? 10000
 10000  100  10000  99.99999237060547
? 0
Ok

1000 '****** STOP ******
1010 OPEN "I",1,"B:BECQUER"
1020 IF EOF(1) THEN END
1025 LINE INPUT#1,A$
1030 IF LEFT$(A$,4)="Pero" THEN STOP:PRINT A$
1045 GOTO 1020
RUN <CR>
Break in 1030
CONT <CR>
Pero aquellas que el vuelo refrenaban
Break in 1030
CONT <CR>
Pero aquellas cuajadas de rocio
Break in 1030
CONT <CR>
Pero mudo, y absorto, y de rodillas
Ok
```

STR$ (Exp. Num.)

Apócope de "String" = cadena.

Función compilable alfanumérica de argumento numérico.

Esta importantísima función convierte el número Exp. Num. en una cadena alfanumérica que contiene los caracteres correspondientes al número. Si el valor de Exp. Num. es positivo, el primer carácter es siempre un espacio (signo "+" no visible). El argumento puede tomar cualquier signo y tener cualquier precisión.

En cierto modo es la función inversa de la VAL. Ver más adelante.

(Atención: no confundir con STRING$ con la que tiene bastante poco que ver).

STRING$ (Exp. Num., Exp. Alf.)

STRING$ (Exp. Num., Exp. Num.1)

De String = Cadena, conjunto de caracteres alfanuméricos.

Función alfanumérica compilable de argumento mixto.

STRING$ tiene como misión fabricar una cadena alfanumérica formada por una serie de caracteres idénticos.

Su expresión general tiene las dos formas que figuran en cabecera. En ambas, el primer argumento (Exp. Num.) expresa el número de caracteres de la cadena a crear. El segundo argumento indica cuál es el carácter o byte a repetir. Si este segundo argumento es alfanumérico, el byte que se toma es el *primero* (o el único si sólo hay uno) y su es numérico, el byte que se repetirá en la cadena será aquél cuyo código ASCII corresponda a Exp. Num.1.

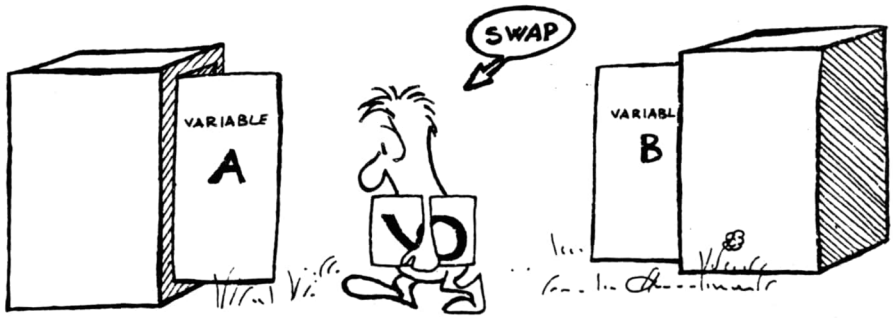
En cualquier caso, los argumentos numéricos pueden tomar cualquier valor comprendido entre 0 (cadena nula) y 255 (Illegal function call si es negativo o superior a 255). En versiones recientes de MSDOS (y sólo trabajando en compilado) el valor de Exp. Num. puede sobrepasar los 255.

```
'******  STR$  ******
A=2125: PRINT STR$(A): PRINT MID$(STR$(A),2) <CR>
 2125
2125
Ok

20 'Una forma de detectar finales de numeros.
30 FOR I=1 TO 50
40 A=RND
50 A$=STR$(A)
60 IF RIGHT$(A$,1)="7" THEN PRINT A$;
70 NEXT I

RUN <CR>
Ok

10 ' *****  STRING$  *****
20 'PROGRAMA SEMEJANTE AL DE SPACE$ PERO QUE RELLENA
30 'CON UN CARACTER CUALQUIERA (ELEGIDO EN EL PROGRAMA)
50 INPUT "Longitud (0=Fin): ",L:IF L=0 THEN END
60 PRINT "Caracter de relleno: ";:C$=INPUT$(1)
70 INPUT "Dato a rellenar: ",D$
80 IF LEN(D$)>L THEN 50
90 DATO$=D$+SPACE$(L-LEN(D$)):PRINT DATO$;LEN(DATO$)
100 GOTO 50
RUN <CR>
Longitud (0=Fin): 22
Caracter de relleno: *
Dato a rellenar: PACORRO
PACORRO**************** 22
Longitud (0=Fin): 22
Caracter de relleno: *
Dato a rellenar: PACO
PACO****************** 22
Longitud (0=Fin): 22
Caracter de relleno: $
Dato a rellenar: A VER QUE PASA
A VER QUE PASA$$$$$$$ 22
Longitud (0=Fin): 0
Ok
```

SWAP Variable 1, Variable 2

"Swap" = "Intercambiar"

Sentencia compilable. Es una fácil y cómoda sentencia con la que intercambiamos los valores contenidos en las dos variables que, separadas por una coma, acompañan al SWAP. La única limitación de esta sentencia es que las dos variables tienen que ser del mismo tipo. En caso contrario resultará un error (Type mismatch) que nos pondrá en nuestro sitio.



Es de hacer notar la rapidez extrema con que se ejecuta esta sentencia, muy apta por tanto para "salvar" dentro de un programa, por ejemplo, los elementos de un vector en otro por tener necesidad de cargar el primero con otros valores sin perder sus valores iniciales.

TAB (Exp. Num.)

De "tabulación", tabulación en pantalla/impresora.

Función compilable. Como le ocurre a SPC(X), con la funcion TAB tampoco podemos cargar variable alguna.

Sirve para que en pantalla o impresora salga lo que esté escrito en el programa a continuación del TAB, debidamente "tabulado", es decir, en la columna que indique el argumento de la función.

Preciso es decir aquí que, si estamos empleando el PRINT USING (o el LPRINT USING), la instrucción en donde se encuentre la función TAB hay que terminarla con punto y coma, ya que el PRINT USING no permite separar el "PRINT" del "USING" ni éste de su máscara, ni ésta de la expresión numérica a sacar.

(Véanse más explicaciones en el capítulo 8 de la primera parte).

```
'******  SWAP  ******
A=111: B=222: SWAP A,B: PRINT A;B
 222  111
Ok
A$="ESTUPENDO": B$="HORRIBLE"
PRINT A$;B$
ESTUPENDOHORRIBLE
Ok
SWAP B$,A$: PRINT A$;B$
HORRIBLEESTUPENDO
Ok

10 DEFINT A-B: DIM A(100),B(100)
30 '
40 PRINT CHR$(7);
50 FOR I=1 TO 100
60 A(I)=I*13
70 B(I)=3*I
80 NEXT I
90 '
100 PRINT CHR$(7);
110 FOR I=1 TO 100
120 SWAP A(I),B(I)
130 NEXT I
140 PRINT CHR$(7);
150 'Son intercambiados los valores de
155 'A() y B() en el tiempo brevisimo
160 'que media entre los dos ultimos pitos.
170 'Si compilaramos este programita
180 'veriamos que el trueque es instantaneo.


'****** TAB ******
PRINT TAB(7);"SIETE"
      SIETE
PRINT TAB(15);"QUINCE"
               QUINCE
PRINT TAB(5);5;TAB(10);10;TAB(15);15
     5    10   15
Ok
```

TAN (Exp. Num.)

De "Tangent", "tangente".

Función numérica compilable de argumento numérico.

Proporciona el valor de la tangente trigonométrica de Exp. Num. siempre en el entendido de que el valor de esta expresión numérica viene dada en radianes.

Esta función, como cualquier otra de naturaleza trigonométrica, sólo tiene aplicación en programas de índole técnica, matemática o científica. TAN, es la inversa de la función ATN (arco(ángulo cuya tangente es el argumento).

TRON y TROFF

Contracciones de "Trace-on" y "Trace-off" que traduciremos (y que nos perdonen los lingüistas) por "Sigueme la traza, el rastro del programa" o "no sigas la traza" respectivamente.

Comando (incluible en programa) no compilable (con reservas que nos resistimos a explicar).

El primero (TRON) sirve como herramienta de rastreo en la detección de posibles errores del programa. Cuando se escribe TRON (CR) (o se incluye en el programa), se activa un mecanismo lógico por el cual, al tiempo que se ejecuta el programa, van apareciendo en pantalla los números de línea —entre corchetes— por los que pasa aquél. De este modo podemos ir siguiendo paso a paso la secuencia de instrucciones del programa y consiguientemente, las vicisitudes por las que atraviesa.

El uso del TRON, junto con sentencias STOP estratégicamente situadas en el programa, posibilitan hacer transparente el completo funcionamiento del mismo al tiempo que nos permiten examinar los valores que van tomando las variables, etc.; en una palabra, seguir paso a paso el programa.

Mientras no escribamos TROFF (CR) persistirá el rastreo y la salida ininterrumpida de líneas entre corchetes. (Por cierto, nunca hemos entendido porqué Microsoft utiliza corchetes como separadores numéricos. Resulta un conglomerado de números apretados, incómodos e ilegibles. El lector exigente puede encontrar el modo de mejorar esto).

```
PRINT TAN(-0.1) <CR>
-.100335

10 ' ****** TAN ******
20 INPUT "ANGULO EN RADIANES: ",ALFA
30 PRINT "TANGENTE DE";ALFA;"=";TAN(ALFA)
35 IF ALFA=0 THEN END ELSE GOTO 20

RUN <CR>
ANGULO EN RADIANES: 1
TANGENTE DE 1 = 1.55741
ANGULO EN RADIANES: 2
TANGENTE DE 2 =-2.18504
ANGULO EN RADIANES: 0
TANGENTE DE 0 = 0
Ok

10 ' ****** TRON - TROFF ******
20 FOR I=80 TO 100: IF I/7=INT(I/7) THEN GOSUB 100
40 NEXT I: END
100 PRINT I "ES DIVISIBLE POR 7":RETURN

RUN <CR>
 84 ES DIVISIBLE POR 7
 91 ES DIVISIBLE POR 7
 98 ES DIVISIBLE POR 7
Ok
TRON <CR>
RUN <CR>
[10][20][30][40][30][40][30][40][30][40][30][100]
84 ES DIVISIBLE POR 7 [40][30][40][30][40][30][40]
[30][40][30][40][30][40][30][100] 91 ES DIVISIBLE POR 7
[40][30][40][30][40][30][40][30][40][30][40][30][40][30]
[100] 98 ES DIVISIBLE POR 7 [40][30][40][30][40][50]
Ok
TROFF: RUN <CR>
 84 ES DIVISIBLE POR 7
 91 ES DIVISIBLE POR 7
 98 ES DIVISIBLE POR 7
Ok
```

USR [Dígito] (Expresión)

Consonantes de "User" (=Usuario).

Función numérica, de uso muy restringido y especializado. Se encarga de llamar a una rutina del usuario escrita en asembler con Expresión como argumento. Con el BASIC de MSDOS (16 bits) el "Dígito" debe estar entre 0 y 9 y se corresponde con el dígito incluido en la sentencia DEF USR para aquella rutina. Si se omite ese dígito, su valor por defecto es el cero.

VAL (Exp. Alf.)

De "Value" (=Valor numérico).

Función numérica compilable de argumento alfanumérico.

Dá el valor numérico de la expresión alfanumérica (argumento).

Si hay espacios en blanco precediendo a la cadena Exp. Alf., éstos no son tenidos en cuenta por la función. Si el argumento de la función es alfabético, la función VAL dará cero. Cuando la cadena-argumento contiene caracteres numéricos y alfabéticos, la función VAL procede de la siguiente forma:

A partir del primer byte de la izquierda, examina los caracteres que va encontrando. Si éstos son numéricos, va preparando con ellos el "valor" de la cadena. Cuando encuentra un carácter no numérico, ya no sigue adelante, incorpora en la función sólo el valor de lo que *por la izquierda* haya podido encontrar como numérico.

Anotemos que la naturaleza "numérica" o "no numérica" de los caracteres no es diáfana ni evidente. Aparte de los 10 dígitos decimales de toda la vida, son también considerados como numéricos, con ciertas reservas:

. + - *E D*

Dada la complejidad que esto puede producir en una función cuya aplicación es por definición obvia y sencilla, no queremos insistir más. Remitimos al lector a los ejemplos de la derecha.

```
10 '******  USR  ******
20 DEF USR0=23000
25 J=2
30 SOL=USR0(J*3.14+1234)
35 'El resultado de la operacion-argumento
36 'va a la rutina USR0, para cargar SOL.
40 PRINT SOL

RUN <CR>
 865.225
Ok

PRINT VAL("1.23");VAL("-233.44"):'***** VAL
 1.23 -233.44
Ok
A$="CATORCE":B$="300 MILLONES"
PRINT VAL(A$);VAL(B$)
 0  300
Ok
A$="ABC333VV":PRINT VAL(RIGHT$(A$,4))
 33
Ok

20 INPUT "CADENA (0 FIN): ",CAD$
22 IF CAD$="FIN" THEN END
30 A#=VAL(CAD$): PRINT A#: GOTO 20
RUN <CR>
CADENA (0 FIN): 12345678
 12345678
CADENA (0 FIN): 10E22
 1D+23
CADENA (0 FIN): 10/2 + 67
 10
CADENA (0 FIN): 123456789012345678 90
 1.2345678901234 57D+19
CADENA (0 FIN): 100E40
Overflow
 1.701411834604693D+38
CADENA (0 FIN):0
Ok
```

VARPTR (Variable)

Contracción de "Variable Pointer" ("Apuntador de Variable").

Función numérica compilable cuyo argumento es el nombre de una variable numérica o alfanumérica, simple o indexada. VARPTR proporciona un número comprendido entre -32768 y 32767 (es decir, un número "entero"). Si el número es negativo, debemos añadirle 65536. De este modo obtenemos en cualquier caso un número comprendido entre 0 y 65535 que se corrresponde con una *dirección física* de la memoria interna donde se encuentra:

El comienzo del valor de la variable si ésta es numérica. Dependiendo del rango de la variable numérica ocupará 2, 4 u 8 bytes y presentará una estructura interna empaquetada de una u otra forma según su rango.

El comienzo del «string descriptor» (descriptor de cadena). Esto no es más que el conjunto de 3 bytes en los que:

El primer byte da la longitud de la cadena-argumento del VARPTR. Este valor estará comprendido (como el de cualquier octeto) entre 0 y 255, longitudes máxima y mínima de la cadena.

El segundo y tercer byte dan, en binario, la dirección real de la memoria interna donde finalmente (y físicamente) se encuentra el contenido de dicho argumento (de la cadena alfanumérica con la que la variable se halla cargada). Es necesario dejar constancia que estos dos bytes están invertidos, es decir, para obtener en nuestro sistema decimal el valor de esa dirección, bastará con que tomemos el valor del segundo byte, lo multipliquemos por 256 y le agreguemos el valor del primero.

Téngase en cuenta que las variables de un programa suelen pasearse frenéticamente por el área de datos de la memoria interna: la función VARPTR dice la verdad del *momento en que* se aplica esa función.

Todo lo anterior puede parecer más o menos confuso. Que quien pretenda trabajar con un BASIC abierto a rutinas en Asembler reconozca que la sentencia que nos ocupa, junto con la POKE, la CALL y la función PEEK, le proporciona una transparencia total de esa misteriosa "area de datos" donde se encuentra el meollo de la memoria interna.

```
10 '******* VARPTR *******
30 INPUT "Dame un dato Alfanumerico (0=FIN): ",D$
40 IF D$="0" THEN END ELSE V=VARPTR(D$)
60 L=PEEK(N)
70 PRINT "La Longitud de ";D$;" es";L: GOTO 30

RUN <CR>
Dame un dato Alfanumerico (0=FIN): MANGO
La Longitud de MANGO es 5
Dame un dato Alfanumerico (0=FIN): MANGULO
La Longitud de MANGULO es 7
Dame un dato Alfanumerico (0=FIN): MANGULORCIO
La Longitud de MANGULORCIO es 11
Dame un dato Alfanumerico (0=FIN): 0
Ok

10 ' ***** Y mas VARPTR *****
20 A%=24
30 A=VARPTR(A%)
35 'La instruccion 40 busca directamente
37 'el contenido de la variale A%
40 PRINT PEEK(A)+255*PEEK(A+1): PRINT "A%=";A%
52 'En la instruccion 60 actuamos directamente
54 'sobre las posiciones de memoria de A%
60 POKE A,25:POKE A+1,25: PRINT "A%=";A%
80 A$="BELLEZA":B$="FEALDAD"
82 'Cambiamos, oh prodigio, FEALDAD en BELLEZA
90 V1=VARPTR(A$):V2=VARPTR(B$)
100 A1=PEEK(V1+1):A2=PEEK(V1+2)
110 PRINT A$,B$
120 POKE V2+1,A1:POKE V2+2,A2
130 PRINT A$,B$

RUN <CR>
 24
A%= 24
A%= 6425
BELLEZA       FEALDAD
BELLEZA       BELLEZA
Ok
```

VARPTR (# Num. Fich.)

Función compilable, de escasísimo uso.

Se trata de otra forma de la función anterior para su utilización con ficheros.

Según sea la naturaleza del fichero (secuencial o de acceso directo) presenta distintos funcionamientos.

En ficheros secuenciales, esta función proporciona la dirección de memoria en la que comienza el buffer (memoria tampón, segmento o trozo de memoria reservado para un fin concreto) que corresponde al fichero secuencial abierto por el canal "Num. Fich.".

En ficheros de acceso directo, nos dá la dirección de la memoria interna donde comienza el buffer que contiene las intocables variables de campo (variables que se definen en la instrucción o intrucciones "FIELD").

WAIT Nro. Puerta, Exp. Num. [,Exp. Num1.]

"Wait" (=esperar).

Sentencia compilable de uso muy especializado.

Su presencia hace suspender la ejecución del programa en tanto no se producen unas determinadas condiciones de entrada de datos del exterior por la puerta cuyo número acompaña a la sentencia.

El dato que entra por dicha puerta es sometido a un XOR (OR exclusivo) con el valor Exp. Num.1. (su valor por defecto es 0). Con el resultado de la anterior operación lógica, se realiza un AND con el valor de Exp. Num.

Si el resultado final de estas operaciones es cero, se recomienza el bucle volviéndose a leer la entrada por Nro. Puerta, y así sucesivamente hasta que el resultado final sea distinto de 0, continuándose la ejecución del programa en la instrucción siguiente al WAIT.

No es raro que se produzca un bucle infinito al utilizar esta sentencia del que sólo se podrá salir haciendo uso de la "violencia" de un Restart.

```
10 ' ***** VARPTR(#C) *****
15 '
20 OPEN "I",1,"B:MACHADO"
25 GOSUB 200
30 LINE INPUT #1, A$
35 PRINT A$
55 PRINT
60 CLOSE 1
65 '
70 OPEN "I",1,"A:NERUDA"
80 LINE INPUT #1,A$
90 PRINT A$
100 GOSUB 200
110 CLOSE:END
120 '
200 A=VARPTR(#1)
210 FOR I=A TO A+37
220 B=PEEK(I)
230 PRINT CHR$(B);
240 NEXT I: PRINT: RETURN

RUN <CR>
El ciego sol se estrella
en las duras
El ciego sol se estrella

Puedo escribir los versos mas tristes esta noche.
Puedo escribir los versos mas tristes
Ok


10 '****** WAIT ******
20 WAIT 3,27,4
30 'El programa suspende su ejecucion hasta que
40 'el valor entrado por la puerta 3 (sea X)
50 'cumple con la expresion:
60 '(X XOR 4) AND 27 <> 0
70 '
```

WEND

Contracción de "While end" ("Final del Mientras que").

Sentencia compilable.

Con ella se cierra el lazo abierto con el WHILE.

Ver la sentencia WHILE, justo a continuación.

WHILE Expresión

"While" = "Mientras que".

Sentencia compilable.

Origina la ejecución de una serie de instrucciones (todas las que haya entre ella y el WEND, final del bucle) siempre y cuando se mantenga *cierta* la "Expresión" que acompaña al WHILE.

Si comparamos este lazo con nuestro bien conocido FOR-NEXT, vemos que en este último obligamos a que una serie de instrucciones sean ejecutadas *un número determinado* de veces.

Con WHILE-WEND, en cambio, el número de veces que se ejecutan las instrucciones comprendidas entre el WHILE y el WEND no es fijo sino, depende de que se mantenga o no la certeza de la expresión inicial. En base a este criterio debe elegirse uno u otro tipo de ciclo.

Como acontecía con el inefable FOR-NEXT, los bucles WHILE-WEND pueden anidarse de una forma semejante a como la hacía aquél, es decir, concéntricamente, cada WEND emparejándose con el WHILE anterior más cercano.

Naturalmente, el número de sentencias WHILE debe coincidir en un programa con el de sentencias WEND. De otro modo nos saltará uno de los errores «WHILE without WEND» o «WEND without WHILE". Como de costumbre, otra vez son Romeo y Julieta que se buscan con encomiable fidelidad.

(Rogamos al ambicioso e infatigable lector que desee ampliar sus conocimientos en este gentil bucle, vuelva grupas al capítulo 7 de la primera parte).

```
10 ' ******* WHILE - WEND ******
20 DIM A(6)
30 'CON ESTE PROGRAMA CONTAMOS 6 VECES
40 'LA CANTIDAD DE NUMEROS ALEATORIOS QUE SE TIENEN QUE
50 'GENERAR PARA ENCONTRAR UNO QUE SEA SUPERIOR A 0.999
60 '
70 WHILE A<.999
80 A=RND:N=N+1
90 WEND
100 '
110 C=C+1:IF C>6 THEN 150
120 A(C)=N
130 A=0:N=0:GOTO 70
140 '
150 FOR I=1 TO 6
160 PRINT A(I);
170 NEXT I

RUN <CR>
 387  1049  230  1846  572  181
Ok

10 ' Seguimos con bucle WHILE - WEND (otro).
12 '
20 A$="MEQUETREFE"
30 WHILE LEFT$(A$,1)<>"Z" AND RIGHT$(A$,1)<>"Z"
40 INPUT A$
50 WEND
60 PRINT "POR FIN!: ";A$;" EMPIEZA Y ACABA EN Z"

RUN <CR>
?ANTONIO
?JUAN
?PEREZ
?GOMEZ
?ZARZAPARRILLA (CACA-CALA)
?GFHGFHFHGFFHG
?ZURRONEZ
POR FIN!: ZURRONEZ EMPIEZA Y ACABA EN Z
Ok
```

WIDTH [LPRINT] Exp. Num.

"Anchura" (pantalla o impresora).

Sentencia compilable que no hace más que preparar a la pantalla o a la impresora para aceptar salidas con un determinado ancho (número de columnas).

El valor de Exp. Num. debe estar comprendido entre 0 y 255. Ese valor, más uno, es el número de columnas que una vez cumplimentadas, darán lugar a que automáticamente se efectúe —sin que el programa diga nada, automáticamente— un salto de línea (CHR$(10), LF) más un retorno de el carro (CHR$(13), CR).

Si no se utiliza la opción LPRINT, estamos dirigiéndonos a la pantalla. Y si utilizamos esta opción, es la impresora la que se organiza de acuerdo con los párrafos anteriores.

WRITE [Lista de expresiones]

De "Write" = Escribir, grabar (en este caso, más bien escribir).

Sentencia compilable. Puede sustituir (algunas pocas veces con ventaja) al omnipresente PRINT.

Si no se utiliza la opción (Lista de expresiones) el resultado es un simple salto de línea (exactamente igual al PRINT en solitario).

La lista de expresiones tras el WRITE (cada una separada de la otra por coma o punto y coma, es lo mismo), produce la salida por pantalla de éstas con *comas* incluidas (recuérdese que este mismo caso, con el PRINT, produce la salida de los datos con una separación prefabricada/tabulada).

Los valores numéricos positivos no presentan el espacio inicial que el PRINT produce.

Las cadenas salen a la palestra con el ribete de las comillas incluidas.

```
WIDTH 90:FOR A=1 TO 50: PRINT "A";:NEXT
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
WIDTH 30:FOR A=1 TO 50: PRINT "A";:NEXT
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
Ok

10 A$="Soy de esa raza mora vieja amiga del sol"
20 FOR I=20 TO 24 STEP 2
30 WIDTH I:PRINT:PRINT A$:NEXT
RUN <CR>
Soy de esa raza mora
 vieja amiga del sol

Soy de esa raza mora v
ieja amiga del sol

Soy de esa raza mora vie
ja amiga del sol
Ok


'****** WRITE ******
'Semejante al 'PRINT', solo anotaremos diferencias.
WRITE 222: PRINT 222
222,
 222
Ok
WRITE 1,2,3: PRINT 1,2,3
1,2,3
 1             2             3
Ok
A=10: B=20 :C=30: WRITE A+B,B,C$A: PRINT A+B,B,C$A
30,20,300
 30            20            300
Ok
A$="Charles": WRITE A$:PRINT A$
"Charles"
Charles
Ok
```

WRITE # Nro. Fich., Lista de expresiones

De "Write" = Excribir, grabar (en este caso, más bien grabar).

Sentencia compilable.

Se trata de la sentencia (además de la "PRINT #") con la que se graban registros en un fichero secuencial.

Conociendo el funcionamiento de la sentencia estudiada en la página anterior y el de la "PRINT #", disponemos de información suficiente como para imaginar cómo se comportará el WRITE # .

Brevemente:

La lista de expresiones componen los datos a grabar en el fichero secuencial que previamente habremos abierto con el número —Num. Fich. (por el canal Num. Fich.).

Esta lista de expresiones puede estar constituida por:

— Sólo por expresiones numéricas.
— Sólo por expresiones alfanuméricas.
— Por una mezcla heterogénea de las dos anteriores.

Graba las expresiones anteriores:

— Si son numéricas como tales números.
— Si son alfanuméricas, con las comillas incluidas.

En cualquier caso, los separadores (comas o puntos y comas) son grabados como tales. Sólo se graba un "fin de registro" (pareja de bytes constituida por Retorno de carro + Salto de línea —o lo que es lo mismo CHR$(13) + CHR$(10)) tras la grabación de la última expresión de la lista.

(Nuevamente remitimos al lector a la nota inicial de esta segunda parte referente a tratamiento de ficheros).

```
10 '****** WRITE #C ******
20 OPEN "O", 1, "B:LAMENTOS"
30 '
40 'Grabamos con WRITE #
50 LINE INPUT A$
60 IF A$="" THEN 100
70 WRITE #1, A$
80 GOTO 50
90 '
100 'Grabamos con PRINT # (a continuacion).
110 '(para ver la diferencia con WRITE #).
120 LINE INPUT A$
130 IF A$="" THEN CLOSE 1: GOTO 170
140 PRINT #1, A$
150 GOTO 100
160 '
170 'Leemos lo grabado.
180 OPEN "I", 2, "B:LAMENTOS"
190 IF EOF(2) THEN END
200 LINE INPUT #2, A$
210 PRINT A$
220 GOTO 190
```

```
RUN <CR>                                          ***
Nina morena y agil, el sol que hace las frutas,   ***
el que cuaja los trigos, el que tuerce las algas, ***
1,2,3,4,5                                         ***
<CR>                                              ***
hizo tu cuerpo alegre, tus luminosos ojos         ***
y tu boca que tiene la sonrisa del agua.          ***
1,2,3,4,5                                         ***
<CR>                                              ***
"Nina morena y agil, el sol que hace las frutas,"
"el que cuaja los trigos, el que tuerce las algas,"
"1,2,3,4,5"
hizo tu cuerpo alegre, tus luminosos ojos
y tu boca que tiene la sonrisa del agua.
1,2,3,4,5
Ok
```

XOR

De «Exclusive OR» = «OR Exclusivo».

Operador lógico compilable.

Como el resto de los operadores lógicos, funciona entre dos operandos numéricos constituyendo una operación lógica. Se caracteriza el XOR por el contraste bit a bit de los operandos en el orden en que ambos estén. Dos bits iguales (los dos "1" o los dos "0") producen "0" al ser sometidos al XOR. Y dos bits desiguales (el primero "0" y el segundo "1" o viceversa) dan "1" como resultado de esta operación lógica. Más que XOR debería denominarse Don Contreras-Bis (el genuino Don Contreras era el NOT).

Aunque de empleo mucho más restringido que AND y OR, le sigue a éstos en frecuencia de utilización.

```
'******  XOR  ******
'(El byte '1' es '00110001'
'     y el '2' es '00110010')
'(Suponemos <CR> tras cada instruccion)

PRINT 1 XOR 1
 0
Ok
PRINT 1 XOR 2
 3
Ok
PRINT "1" XOR "1"
Type mismatch
Ok
A=10: B=20: C=30
Ok
PRINT A>B XOR A>C
 0
Ok
PRINT A<B XOR A<C
 0
Ok
PRINT B>A XOR B>C
-1
Ok
PRINT B<C XOR B>A
 0
Ok
PRINT A*B>C XOR A*C<B
-1
Ok
PRINT "ALFONSO">"ANTON" XOR "ANA">"AARONEZ"
-1
Ok
```

ANEXO I

COMPILACION

Se trata de convertir un programa Basic ("Programa Fuente") en otro, en lenguaje máquina ("Programa Objeto"), que produzca los mismos resultados aunque a una velocidad de proceso mucho mayor.

Para compilar un programa confeccionado en Basic hay que tomar en consideración los siguientes puntos:

1.— Formato del programa.
2.— Ficheros y Programas que intervienen.
3.— Sentencias Basic conflictivas o a tener en cuenta.
4.— Otras observaciones importantes.
5.— Operatoria.

1.-- FORMATO DEL PROGRAMA.

El programa a compilar tiene que ser salvado en ASCII. Esto se consigue de los siguientes modos:

- Desde el Basic, con un SAVE "PROGRAMA", **A**
- Desde cualquier Editor.
- Desde cualquier Procesador de Textos, confeccionando el programa como "NO documento".

2.— FICHEROS Y PROGRAMAS QUE INTERVIENEN.

Muy en resumen y simplificando, son éstos como mínimo:

– Por supuesto, el programa a compilar.
– El programa-compilador: BASCOM.COM.
– El programa "linkador": L80.COM (o LINK.COM).
– El programa-módulo: BRUN.COM (o BASRUN.EXE).
– La librería BASLIB.REL.
– El fichero de carga BCLOAD.

(Excepto el primero, el resto forma parte del software ya completamente preparado para su inmediata aplicación por el programador de Basic que vaya a compilar. No hay que **hacer** nada con ellos. Sólo, asegurarse de que están presentes en el disco.)

3.– SENTENCIAS BASIC CONFLICTIVAS.

En extracto son (sólo mencionamos las más importantes):

CHAIN:

Se puede utilizar en su forma simple para encadenar programas, pero el compilador no admite la opción MERGE, ALL, etc. Para transmitir valores de variables de un programa a otro, es necesario emplear la sentencia COMMON, con preferencia dentro de un %INCLUDE (ver un poco más abajo).

COMMON:

Se utiliza exactamente igual que con el Intérprete. Sólo es preciso tener en cuenta que las sentencias COMMON deben aparecer en el programa *antes* que cualquier sentencia *ejecutable.* Y son ejecutables todas las sentencias excepto:

– COMMON
– DEFDBL, DEFSNG, DEFINT, DEFSTR
– DIM
– OPTION BASE
– REM
– % INCLUDE

Para no complicarnos la vida: las COMMON deben ir cuanto antes, al principio del programa. Pero atención: si –caso normal–

en los COMMON hay vectores (o variables indexadas en general), las sentencias DIM han de ir *antes* que las COMMON. Para segurar el paso correcto de variables entre programas, aconsejamos vivamente se haga uso del %INCLUDE (véase).

DIM:

Sólo advertir que las instrucciones DIM deben ir también al principio del programa (y, como acabamos de decir, delante de las COMMON).

DEFXXX:

Incluímos las cuatro definiciones INT, SNG, DBL y STR. También hay que hacerlas aparecer en las primeras líneas del programa.

La razón de ser de esta condición (análoga a la de DIM) es la siguiente:

El compilador trabaja por líneas descendentes y consecutivas. No sigue la lógica del programa en cuanto a GOTOs y GOSUBs. Veamos un ejemplo para aclarar algo que inquieta a los programadores no muy expertos (y a algunos que lo son):

Los programas (por llamarlos de alguna manera):

```
10 GOTO 100
20 A=12345678
30 RETURN
40 '
100 DEFDBL A
110 GOSUB 20
120 PRINT A
```

y también:

```
10 GOTO 100
20 A$(11)="PASCUALA"
40 RETURN
50 REM
100 DIM A$(20)
120 GOSUB 20
130 PRINT A$(11)
```

Ejecutados en intérprete (pruébense), funcionan divinamente. El primero sacará en pantalla un orondo '12345678' y el segundo una "PASCUALA" igualmente aceptable. Vamos ahora a compilarlos. El primero se dejará, pero luego al ejecutarlo, nos sacará el valor de A en coma flotante (1.2345E+07), mientras que el segundo, ni siquiera se podrá compilar porque nos dará dos errores fatales: "Suscript out of range" y "Array already dimensioned" durante la compilación. Y ello es porque la línea 20 se compila sin tener en cuenta el camino lógico del programa, dando lugar a los desastres anteriores.

ERASE:

Sentencia absolutamente prohibida en compilación. El espacio reservado para vectores es sagrado. No se puede andar jugando con él cuando un programa está hecho para ser compilado.

%INCLUDE:

Sentencia-comando nuevo para quien nunca haya trabajado con el compiler. Sólo utilizable en un programa que va a compilarse. Si se intenta ejecutar en intérprete saldrá un ingenuo Sintax error.

Esta útil sentencia-comando cuyo formato más general es:

%INCLUDE Nombre de Programa en Basic

tiene la virtud de que cuando la encuentra el compilador, busca el programa con ese nombre y lo incorpora al principal como si de un MERGE se tratase. (El 'Nombre del Programa' se escribirá sin comillas.)

Esto nos permite, por ejemplo, confeccionar un programa en el que se incluyen todas las instrucciones que sean comunes a dos o más programas principales que se encadenen (con CHAIN) en la aplicación. Y hablando de comunes, son precisamente las COMMON las que más oportunamente están pintiparadas para formar parte del programa que va junto al %INCLUDE. Es la mejor forma de asegurar la identidad de las COMMON en uno y otro programa. Las DIM también pueden (deberı) incluirse en este programa auxiliar (no da error de 'redimensionado' cuando el control pasa de uno a otro programa mediante CHAINs.)

Notas importantes relacionadas con el INCLUDE:

1.— El subprograma que corresponde al INCLUDE debe ser salvado en ASCII (igual que el principal, véase en este mismo anexo el apartado "1.— FORMATO DEL PROGRAMA".).

2.— El programa del INCLUDE va a *entrar* en el principal al compilarse por lo que el número de línea más bajo de este subprograma tendrá que ser *superior* al de la línea en la que va el INCLUDE en el principal. Análogamente, la línea más alta en el subprograma del INCLUDE debe ser *inferior* a la siguiente del INCLUDE en el principal.

3.— No se pueden anidar INCLUDEs dentro de uno de estos subprogramas.

4.— La sentencia-comando INCLUDE o bien está sola en una línea del programa principal o es la *última* de las instrucciones contenidas en su línea. Por ejemplo:

```
20 DEFDBL N:PITO$=CHR$ (7):%INCLUDE COMUNES1
```

5.— Las referencias que haya en el subprograma (GOTO, GOSUB, etc.) a líneas tanto de éste como del programa principal, son perfectamente permisibles siempre que, naturalmente, tales líneas *existan.*

Ejemplo general:

Programa principal:

```
10 'Empleo del INCLUDE.
20 %INCLUDE B:COMUNES
70 GOTO 2000
80 %INCLUDE RUTINAS
2000 'Empieza programa especifico.
2010 BLA BLA BLA
2020 MAS BLA BLA
- - - -
```

Subprograma 'COMUNES'

```
30 DEFINT A-G:DEFSTR S
40 DIM SE$(7),ME$(12),MES(13),X$(8)
50 COMMON SE$(),ME$(),MES()
```

Subprograma 'RUTINAS'

```
100 'Rutina 1
110 PRINT "FENOMENO":RETURN
120 'Rutina 2

- - - - -
1800 'Rutina 314
- - - - -
1990 REM Fin de Rutinas.
```

ON ERROR GOTO:

Recomendamos que no se use esta sentencia. En rigor no la prohibe el compilador. Exige un formato muy especial (véase más adelante) y, sobre todo, da lugar a un consumo extra muy considerable de memoria. Por otra parte, en un programa de calidad, siempre se puede encontrar una solución alternativa más elegante que la que lleva a cabo una rutina de control de errores para salir de una situación anómala.

REM:

Esta sentencia (o su equivalente, la comilla simple) se la salta olímpicamente el compilador. Por consiguiente, todas las aclaraciones, notas y comentarios que se quieran incorporar a un programa que va a ser compilado, no consumen ni un solo byte. Permanecen intactas, por supuesto, en el programa fuente.

Esto invita a ser generosos a la hora de documentar un programa:

```
10 'Este programa es, verdaderamente, la monda, porque
20 'ademas de conseguir que la parte primera de la se-
30 'gunda bla bla bla bla bla (350 blas).
40 '
- - - -
500 'Acaban las explicaciones y empieza el programa.
510 '
520 INPUT "Obsequiame con 2 numeros A y B: ",A,B
530 PRINT "GRACIAS"
540 'Hay que tener en cuenta que bla, bla...
- - - -
```

Dependiendo de la locuacidad del programador, este programa, en intérprete, puede dejar exhausta la memoria interna a base de comentarios. El compilador, en cambio, sólo tomaría las líneas 520 y 530, únicas que no contienen observaciones.

Interesante es también anotar que tampoco influye en la ocupación de un programa compilado, ni la cantidad de líneas de informaciones (ni tampoco la de líneas físicas) ni la longitud de los nombres de las variables que puedan intervenir en el programa.

4.—OTRAS OBSERVACIONES IMPORTANTES.

- Siempre que sea posible se aconseja la utilización de variables enteras. Esto se traduce por un aumento muy considerable en la velocidad de ejecución de las instrucciones, especialmente en las cíclicas.
- El compilador contempla el empleo de constantes y variables de doble precisión en operaciones trascendentes como LOG, SIN, ATN, etc.
- Cuestión de la mayor importancia es la disciplina férrea que el compilador exige en el uso apropiado de los niveles numéricos de precisión. Aquí tenemos que extendernos un tanto.

  Por de pronto, cuando en una expresión se presentan variables (u operandos, en general) con diferentes niveles de precisión, el compilador los trata como si fueran todos de la precisión *más alta,* presente en la expresión.

  Por ejemplo, en A=B# + C! + D%, las variables C! y D% se convierten en variables de doble precisión por el hecho de existir B# (aunque, el valor final se carga en A con la precisión que 'A' tenga, simple precisión si no se especifica otra).

  La cosa no parece tener miga, ¿verdad?. Pues la tiene. Porque si los valores de la expresión (parte derecha) son *enteros,* se puede armar una buena. Y los tales valores (léase a cámara lentísima la perogrullada que sigue) son enteros mientras no dejen de serlo. Otra vez: son enteros mientras no se nos diga que no lo son. Y mientras puedan serlo lo son. El enredijo anterior sólo puede aclararlo un ejemplo. .

```
10 A=100*100*100#:PRINT A
30 A=100!*100!*100!:PRINT A
50 A#=100!*100!*100!:PRINT A#
90 A#=100*100!*100#:PRINT A#
110 A=100*100*100:PRINT A
120 A#=100*ASC("d")*ASC("d"):PRINT A#
```

Sabiendo que el ASCII de 'd' vale 100, si ejecutamos este programita en intérprete, obtendremos:

```
1E+07
1E+07
1000000
1000000
1E+07
1000000
```

Ciertamente, nada que objetar. Ahora, lo compilamos. Bueno, eso quisiéramos. No podemos, no nos deja el BASCOM. Vamos a ver qué ocurre. En cada una de las expresiones aritméticas aparece el '100' de una u otra forma. Cuando el 100 tiene adosado un '!' o un '#' los "cienes" son reales y de simple y doble precisión respectivamente. Pero cuando aparece como un desnudo '100', sin aditamento, el compilador entiende que es un 100 con precisión de "entero". Y cuando en la expresión hay otro 100 de mayor precisión, todo va bien pues prevalece la precisión más alta como antes hemos dicho. Pero resulta que en la línea 110, los 3 "cienes" aparecen como enteros. Entonces el compilador entiende que se intenta obtener otro "entero" (no es capaz de saltar a otra precisión superior) y claro, el millón resultante no cabe en un "entero" y la compilación produce un error fatal de overflow.

Aceptamos de mala gana esa travesura del compilador. Eliminamos la línea 110. Volvemos a compilar. Ahora no hay errores. Pero sí un horror. He aquí el resultado de la ejecución del programa compilado:

```
1E+07
1E+07
1000000
1000000
1E+07
16960
```

No vamos a meternos en cómo la línea 120 (que ha sido capaz de superar la prueba de la compilación) puede hacer que 1000000 se convierta en 16960. Es la consecuencia de una "operación proscrita": "enteros" que sobrepasan su límite admisible (hasta 32767).

Un caso típico en que se produce este horror es el de un proceso de transformación de clave como éste:

```
330 CL=ASC(CL$)*1000+ASC(MID$(CL$,2))*100
      +ASC(MID$(CL$,3))*10+ASC(MID$(CL$,4))
```

Todos los operandos de la instrucción son "enteros". Es seguro que el resultado de la operación sobrepasará el límite de los enteros. No habrá error pero el valor de CL será impredecible.

Para salir de esta situación basta con "decir" a uno cualquiera de los operandos que es un número real. Por ejemplo:

```
330 CL=ASC(CL$)*1000!+ASC(MID$(CL$,2))*100
      +ASC(MID$(CL$,3))*10+ASC(MID$(CL$,4))
```

con lo que automáticamente, la expresión completa se convierte en "real" y CL queda cargado con su valor correcto.

Otro caso sangrante (ahora es el compilador el que lo hace bien, y el intérprete el que lo hace "mal") es el siguiente:

```
100 PRINT SQR (2500)
```

Obtendremos, claro está, 50.

(La compilación va a producir el mismo resultado).

Pero si ponemos:

```
100 PRINT INT (SQR (2500) )
```

El resultado es (Dios mío), 49.

Compilando, el resultado será el correcto: 50.

No hay que hacer un drama de esto. Pero hay que saberlo. Hay que tener cuidado con la parte entera de un número cuando se uti-

lizan exponentes, raíces, etc., por esa torpe tendencia de la "calculadora" a pretender que la raíz cuadrada de 2500 vale 49.9999655. (Este caso, ya un poco rebuscado, se arreglaría poniendo, un tanto chapuceramente, INT (SQR (2500)+.001))

5.–OPERATORIA:

Una vez que el programa a compilar ha sido salvado en ASCII, de acuerdo con el punto1, se sitúa el programador en modo control de sistema, abandonando al intérprete si en él estuviera. Entonces se ejecuta el programa compilador escribiendo:

BASCOM Nombre prog. compilado = Nombre progr. en intérprete

Por ejemplo:

```
A>BASCOM A:NOMINA1=A:NOMINA1
```

O, lo que produciría idéntico resultado (cuando el nombre que queremos para el programa compilado coincide con el del nombre en Basic):

```
A>BASCOM =A:NOMINA1
```

Si no queremos propiamente compilar sino utilizar el compilador como simple "detectador de errores", pondremos:

```
A>BASCOM ,=NOMINA1
```

En cualquiera de los casos anteriores, y durante el tiempo de la compilación (o de la pseudocompilación en el último caso), van apareciendo (si los hay), los errores detectables por el compilador en forma codificada e indicando el lugar exacto en que el error se encuentra.

Los errores producidos pueden ser de dos tipos:

- Errores de advertencia y
- Errores fatales.

Sólo los segundos hacen imposible el conexionado final (y la producción en definitiva del programa en lenguaje máquina directamente ejecutable).

El error de advertencia ("venial") más típico es el 'ND' (error de "no dimensionado"). Brota con cada referencia a un vector que no haya pasado por un previo DIM. Si más tarde, al ejecutarlo, el programa se encuentra con un elemento de un vector no dimensionado cuyo índice sobrepasa el '10', se producirá indefectiblemente un:

"Internal error. String space corrupt"

Los errores fatales ("mortales") obligan a volver al intérprete (o al proceso de textos) para desfacer los entuertos que en el programa en ASCII hubiere. Ello efectuado, se vuelve a compilar (de uno a tres minutos).

Los "errores fatales" clásicos son los siguientes:

DD: vector dimensionado dos veces.
FN: Error FOR/NEXT (normalmente NEXT sin FOR).
LL: Línea demasiado larga (más de 254 caracteres).
SN: Error de sintáxis, de detección ordinariamente fácil.
TM; Type mismatch (alfan. y/o numer. desencajados).
UF: Función no definida.

Una vez eliminados los errores y llegados a un final feliz de esta fase, veremos aparecer el mensaje "0000 Fatal Errors". El BASCOM (Basic compiler) ha producido un fichero-programa reubicable y conexionable ("relocatable and linkable file"). Este programa (que queda grabado en el disco con el apéndice .REL cuando se trabaja bajo CP/M y .OBJ cuando estamos en sistema operativo MSDOS) no es todavía inmediatamente utilizable por el usuario. Es preciso "linkarlo" con módulos de librería y rutinas en ensamblador para llegar al programa en lenguaje máquina final. Este proceso es totalmente automático y corre a cargo del programa llamado 'L80' (CP/M) o 'LINK' (MSDOS):

L80 Progr. salido del BASCOM, Progr. Final/N/E

En el ejemplo anterior:

A>L80 NOMINA1,NOMINA1/N/E

O, lo que es lo mismo:

```
A>L80 NOMINA1 /N/E
```

Si queremos que el programa final se llame NOM1, escribiremos:

```
A>L80 NOMINA1,NOM1/N/E
```

Obsérvese que el orden de los programas está invertido respecto al criterio seguido en el BASCOM.

Tras un tiempo no mayor de tres minutos, obtenemos finalmente un programa ejecutable en lenguaje máquina con todas las ventajas que ello comporta.

ANEXO II

ERRORES EN BASIC MICROSOFT

(El asterisco (*) que figura en algunos mensajes de error, significa que se trata de un error no detectable por el compilador, por lo que es sumamente importante procurar que el error no tenga lugar. Es decir, en la compilación se produce un filtraje de una gran parte de los errores por lo que el compilador actúa de buena ayuda para el programador.)

Mensaje en Pantalla	**Cod. y Explicación del error.**
Bad file mode (*)	54 Se ha pretendido operar con un fichero (GET, PUT, INPUT #, etc.) que no ha sido abierto. O se intenta un GET o PUT en un fichero secuencial. O se ha utilizado con el OPEN un distintivo diferente a "R", "O" o "I".
Bad file number	52 Se intenta abrir un fichero con un número superior al rango permitido (3 si nada se ha especificado al cargar el intérprete Basic).
Bad file name (*)	64 Se ha especificado un nombre de fichero no legal. Por ejemplo el número de caracteres del nombre es superior a 8.
Bad record number (*)	43 El número de registro a leer o grabar no está entre los límites permitidos (es inferior a 1 o superior a 32767). El caso típico de comisión de este error es el

		de que el valor de la variable numérica utilizada con el GET o PUT es cero.
Can't continue	17	La ejecución del programa no puede continuar. Se ha pretendido seguir la ejecución de un programa que se ha autoabortado por culpa de un error. O ha sido modificado (el programa) después de una interrupción. Se puede continuar, sin embargo, con un GOTO NN. También se puede efectuar un RUN NN, pero en este caso todas las variables numéricas son inicializadas a cero y las alfanuméricas a Nulls.
Disk full (*)	61	El disco está completamente lleno, no queda espacio disponible.
Disk I/O error	57	Error de disco. Puede ocurrir en lecturas o grabaciones. El disco está físicamente mal. (Ordinariamente este error no surge ya que se le adelanta el del Sistema Operativo, por ejemplo, en CP/M, el estremecedor "Bdos error").
Division by zero (*)	11	El divisor de un quebrado es cero, lo cual aritméticamente es un infinito al que no llega la máquina.
Duplicate definition	10	Un vector ya dimensionado, se ha pretendido dimensionar nuevamente.
Field overflow (*)	50	Se intenta meter en una variable de campo, una cadena alfanumérica con un número de caracteres superior al especificado en la definición del FIELD.
File already exists	58	El fichero ya existe. Este error surge con un NAME (cambio de nombre de una file), cuando el nuevo nombre es idéntico a otro que ya existe en el directorio.
File already open (*)	55	Se ha pretendido abrir un fichero con un número de canal ya utilizado (y abierto) por ese u otro fichero.

File not found (*)	56	Se intenta cargar (LOAD) o ejecutar (RUN) un programa que no existe. O se ha querido hacer un KILL de una file que no está en el directorio. O se ha intentado un OPEN "I" (lectura) de un fichero inexistente.
FOR without NEXT	26	En el programa hay un FOR que no encuentra su pareja NEXT. (Es más frecuente que el error que se produce con estos ciclos sea el NEXT without FOR —véase— ya que el programa mantiene la esperanza de que aparezca su ansiado NEXT).
Illegal direct	12	Se ha querido ejecutar en modo directo una instrucción que sólo puede funcionar dentro de un programa (por ejemplo un DEF FN).
Illegal function call (*)	5	Un parámetro de una función está fuera de los límites permisibles. Por ejemplo: la longitud especificada en un LEFT$, MID$, RIGHT$, no está comprendida entre 0 y 255. Otro ejemplo (quizá el más típico) es el de que en un MID$ (A$,X,Y), el valor de X es cero.
Input past end (*)	62	Se ha intentado seguir leyendo en un fichero secuencial (con un INPUT #), más allá de su contenido. O el fichero secuencial está vacío. (Este error se evita con el uso del detectador de fin de fichero EOF (F).)
Internal error (*)	51	Error interno. El caso más frecuente tiene lugar en programas compilados, cuando en un vector, se ha pretendido utilizar un elemento por encima de la dimensión de aquel. El área de datos queda desbaratada y el mensaje anterior aparece con la apostilla "String space corrupt" (que, como se ve, no es algo tan grave como sugiere el mensaje).

Line buffer overflow	23	Se ha pasado el programador en el número de caracteres permitido para una línea de programa (254 en compilado y 255 sin compilar).
Missing operand	22	Falta el operando. Una expresión contiene un operador al que le falta un operando.
NEXT without FOR	1	NEXT sin FOR. (Seguramente en el NEXT se ha puesto otra variable distinta.)
No RESUME	19	Falta un RESUME en una rutina de tratamiento de errores.
Out of data (*)	4	Nos hemos quedado sin datos en una instrucción READ, bien porque hayamos olvidado una línea de DATA o porque no hayamos puesto un RESTORE.
Out of memory (*)	7	Se nos ha acabado la memoria interna. No hay espacio suficiente, por exceso de variables y/o programa demasiado largo. Se puede remediar utilizando la sentencia ERASE (no muy recomendable ya que el ERASE lo tiene proscrito el compilador).
Out of string space	14	No hay sitio para las cadenas. Se ha agotado el espacio de memoria asignado para las cadenas alfanuméricas.
Overflow (*)	6	Exceso en la asignación de una variable numérica. El caso más frecuente es el de asignar a una variable entera un valor superior a 32767 o inferior a −32768.
Resume without error	20	Se ha puesto en el programa una sentencia RESUME sin error.
RETURN without GOSUB	3	Se ha entrado en un subprograma sin el correspondiente GOSUB, bien porque se haya utilizado un GOTO en su lugar o simplemente porque no se haya pro-

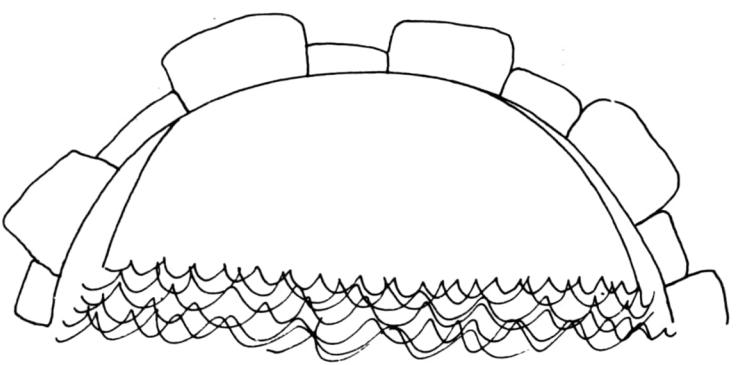
		tegido debidamente el comienzo del subprograma.
String fórmula too complex	16	Una expresión aritmética o de cadena es demasiado compleja. Hay que descomponerla en expresiones más cortas.
String too long (*)	15	Cadena demasiado larga. La longitud máxima de una cadena es de 255 caracteres.
Subscript out of range	9	Se ha intentado emplear un elemento de vector superior al designado en su DIM (o superior a 10 si no se ha dimensionado). Atención a la nota que respecto a la compilación hemos puesto en el "Internal error".
Syntax Error	2	Error de sintáxis. Cualquier cosa escrita en el programa ininteligible para el intérprete. (El caso más frecuente es el de ausencia de un paréntesis y el de una mala puntuación.)
Type mismatch (*)	13	Desacuerdo entre constantes o variables numéricas y alfanuméricas. Se ha dado como numérico algo alfanumérico o viceversa.
Undefined line	8	Número de línea inexistente en el programa a la que se hace referencia con un GOTO, ON, etc.
Undefined user function	17	Se está queriendo utilizar una función que no ha sido definida anteriormente por el programador.

ANEXO III

TABLA DE CODIGOS ASCII

0	CONTROL –@	25	CONTROL –Y
1	CONTROL –A	26	CONTROL –Z
2	CONTROL –B	27	CONTROL –[(ESC)
3	CONTROL –C	28	CONTROL –\
4	CONTROL –D	29	CONTROL –]
5	CONTROL –E	30	CONTROL –^
6	CONTROL –F	31	CONTROL – ¯
7	CONTROL –G (BEL)	32	ESPACIO (SPC)
8	CONTROL –H (BS)	33	!
9	CONTROL –I (TAB)	34	"
10	CONTROL –J (LF)	35	#
11	CONTROL –K	36	$
12	CONTROL –L	37	%
13	CONTROL –M (CR)	38	&
14	CONTROL –N	39	'
15	CONTROL –O	40	(
16	CONTROL –P	41	)
17	CONTROL –Q	42	*
18	CONTROL –R	43	+
19	CONTROL –S	44	,
20	CONTROL –T	45	–
21	CONTROL –U	46	.
22	CONTROL –V	47	/
23	CONTROL –W	48	0
24	CONTROL –X	49	1

50	2	70	F	90	Z	110	n
51	3	71	G	91	[	111	o
52	4	72	H	92	\	112	p
53	5	73	I	93	]	113	q
54	6	74	J	94	^	114	r
55	7	75	K	95	_	115	s
56	8	76	L	96		116	t
57	9	77	M	97	a	117	u
58	:	78	N	98	b	118	v
59	;	79	O	99	c	119	w
60	<	80	P	100	d	120	x
61	=	81	Q	101	e	121	y
62	>	82	R	102	f	122	z
63	?	83	S	103	g	123	{
64	@	84	T	104	h	124	|
65	A	85	U	105	i	125	}
66	B	86	V	106	j	126	~
67	C	87	W	107	k	127	DEL
68	D	88	X	108	l		
69	E	89	Y	109	m		



INDICE

Pág.

Preámbulo 9

PARTE PRIMERA. PROGRAMACION ELEMENTAL

Capítulo 1.— FUNDAMENTOS DEL BASIC (I) 13
Los "Basic" y el Basic de Microsoft 14
Por Defecto 14
El Teclado 15
Puesta en Marcha 17
Modos de Operación 19
Definiciones 22

Capítulo 2.— FUNDAMENTOS DEL BASIC (II) 25
Constantes y Variables 25
Vectores y Matrices 29
Operadores 33
Compilación e Interpretación 36
Numeración de Líneas de un programa 38
Corrección de un programa 39
Caracteres de Control 44

Capítulo 3.— COMANDOS, SENTENCIAS, ASIGNACIONES 45
Primeros comandos y sentencias 45
Asignación de Variables:
Asignación directa 50
Asignación a través de READ-DATA 56
Asignación a través de INPUT 60
Puntuación 62

Capítulo 4.— BIFURCACIONES Y CONDICIONES 69

Capítulo 5.— FUNCIONES 87
Manejo de constantes y variables alfa 87
Operaciones con cadenas 93
Otras funciones de cadena 96
Codificación ASCII y sus funciones 100

Capítulo 6.— FUNCIONES PARA TODO 107
Funciones Numéricas 107
Otras funciones de cadena 113
Resumen de las funciones del BASIC 117
Funciones definidas por el programador 117

Capítulo 7.— BUCLES Y NIDOS 123

Capítulo 8.— LA PANTALLA 137
Instrucciones de Pantalla 137
Uso de coordenadas completas 143
Secuencias de escape 144
Crítica de la sentencia INPUT 148

PARTE SEGUNDA: MANUAL DE BASIC SISTEMATICO

Organización y modo de uso de este manual 153

ANEXOS: I.— Compilación 299
II.— Errores en Basic Microsoft 311
III.— Tabla de Códigos ASCII 317